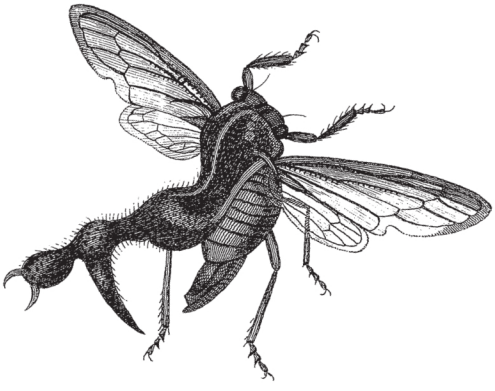
What You Need to Know About Oracle Database Architecture and Features

5th Edition
Covers Oracle Database 12c and Earlier Releases

Oracle Essentials

Oracle Database 12c



Rick Greenwald, Robert Stackowiak & Jonathan Stern

O'REILLY®

Oracle Essentials

Written by Oracle insiders, this indispensable guide distills an enormous amount of information about the Oracle Database into one compact volume. Ideal for novice and experienced DBAs, developers, managers, and users, *Oracle Essentials* walks you through technologies and features in Oracle's product line, including its architecture, data structures, networking, concurrency, and tuning.

Complete with illustrations and helpful hints, this fifth edition provides a valuable one-stop overview of Oracle Database 12*c*, including an introduction to Oracle and cloud computing. *Oracle Essentials* provides the conceptual background you need to understand how Oracle truly works.

Topics include:

- A complete overview of Oracle databases and data stores, and Fusion Middleware products and features
- Core concepts and structures in Oracle's architecture, including pluggable databases
- Oracle objects and the various datatypes Oracle supports
- System and database management, including Oracle Enterprise Manager 12*c*
- Security options, basic auditing capabilities, and options for meeting compliance needs
- Performance characteristics of disk, memory, and CPU tuning
- Basic principles of multiuser concurrency
- Oracle's online transaction processing (OLTP)
- Data warehouses, Big Data, and Oracle's business intelligence tools
- Backup and recovery, and high availability and failover solutions

"Oracle Essentials gives a clear explanation of the key database concepts and architecture underlying the Oracle database. It's a great reference for anyone doing development or management of Oracle databases."

—Andrew Mendelsohn
Senior Vice President, Database Server Technologies, Oracle Corporation

Rick Greenwald, Director of Product Management at Oracle Corporation and responsible for the Oracle Database Cloud, is the principal author of 19 technical books, including previous editions of *Oracle Essentials*.

Robert Stackowiak, Vice President of Information Architecture and Big Data in Oracle's Enterprise Solutions Group, has more than 25 years of experience in database architecture and software development.

US $39.99 CAN $41.99
ISBN: 978-1-449-34303-3

Twitter: @oreillymedia
facebook.com/oreilly

O'REILLY®
oreilly.com

FIFTH EDITION

Oracle Essentials

Rick Greenwald, Robert Stackowiak, and Jonathan Stern

O'REILLY®
Beijing • Cambridge • Farnham • Köln • Sebastopol • Tokyo

Oracle Essentials, Fifth Edition

by Rick Greenwald, Robert Stackowiak, and Jonathan Stern

Printed in the United States of America.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (*http://my.safaribooksonline.com*). For more information, contact our corporate/institutional sales department: 800-998-9938 or *corporate@oreilly.com*.

Editor: Ann Spencer
Production Editor: Christopher Hearse
Copyeditor: Kiel Van Horn
Proofreader: Rachel Leach

Indexer: Lucie Haskins
Cover Designer: Randy Comer
Interior Designer: David Futato
Illustrator: Rebecca Demarest

September 2013: Fifth Edition

Revision History for the Fifth Edition:

2013-09-04: First release

See *http://oreilly.com/catalog/errata.csp?isbn=9781449343033* for release details.

ISBN: 978-1-449-34303-3

[LSI]

Table of Contents

Preface

We dedicate this book to the memory of one of our original coauthors, Jonathan Stern. Jonathan unexpectedly passed away in March of 2007. Yet his memory lives on for those of us who knew him and, in many ways, for those who will read this book. Let us explain.

The original outline for this book was first assembled at the ubiquitous coffee shop located in the Sears Tower in Chicago. It was 1998 and the authors had gathered there with a common goal. We were all Oracle employees working in technical sales roles and had visited many organizations and companies. We found that many IT managers, Oracle Database Administrators (DBAs), and Oracle developers were quite adept at reading Oracle's documentation, but seemed to be missing an understanding of the overall Oracle footprint and how to practically apply what they were reading. It was as if they had a recipe book, but were unclear on how to gather the right ingredients and mix them together successfully. This bothered all of us, but it particularly frustrated Jonathan.

Jonathan was the kind of person who sought to understand how things worked. Nothing delighted Jonathan more than gaining such an understanding, then spending hours thinking of ways to translate his understanding into something that would be more meaningful to others. He believed that a key role for himself while at Oracle was the transfer of such knowledge to others. He continued to perform similar roles later at other companies at which he worked.

Writing the first edition of *Oracle Essentials* was a lengthy process. Jonathan wrote several of the original chapters, and he also reviewed some of the other original work and was quick to identify where he thought something was wrong. For Jonathan, "wrong" meant that the text could be misinterpreted and that further clarity was needed to make sure the right conclusion was drawn. The first edition became much more useful through Jonathan's efforts. He was always quite proud of that effort. Even as the book changed with succeeding editions and Jonathan moved on to other companies, he continued to feel that this book remained an important accomplishment in his life.

Some explanations of how Oracle works are fundamental to the database and have not changed in subsequent editions of the book, so some of Jonathan's original work remains here, although much of the surrounding text is now considerably different. Of course, some entire sections describing the complex steps that were once needed to manage and deploy older releases of the database are no longer relevant and thus are no longer included. Jonathan would probably view Oracle's self-managing, self-tuning, and cloud-enabling improvements as incredible achievements, but would also wonder whether it is a good thing that people can know even less today about how the database works but still deploy it.

So, we introduce you to the fifth edition of *Oracle Essentials*. We have made many changes in this edition. Some, of course, result from changes in features in Oracle Database 12*c* and the ways that you can now use and deploy the latest release of the database. But we have also made a considerable effort to go back and rewrite parts of the book that we did not believe possessed the clarity needed by our readers—clarity that Jonathan would want in such a book. So, he influences us still.

Goals of This Book

Our main goal is to give you a foundation for using the Oracle Database effectively and efficiently. Therefore, we wrote with these principles in mind:

Focus
: We've tried to concentrate on the most important Oracle issues. Every topic provides a comprehensive but concise discussion of how Oracle handles an issue and the repercussions of that action.

Brevity
: One of the first decisions we made was to concentrate on principles rather than syntax. There simply isn't room for myriad syntax diagrams and examples in this book.

Uniqueness
: We've tried to make this an ideal first Oracle book for a wide spectrum of Oracle users—but not the last! You will very likely have to refer to Oracle documentation or other, more specific books for more details about using Oracle. However, we hope this book will act as an accelerator for you. Using the foundation you get from this book, you can take detailed information from other sources and put it to the best use.

This book is the result of more than 65 combined years of experience with Oracle and other databases. We hope you'll benefit from that experience.

Audience for This Book

We wrote this book for people possessing all levels of Oracle expertise. Our target audiences include DBAs who spend most of their workday managing Oracle, application developers who build their systems on the data available in an Oracle Database, and system administrators who are concerned with how Oracle will affect their computing environments. Of course, IT managers and business users interact more peripherally with the actual Oracle Database, but can still gain from a better understanding of the product. On the one hand, anticipating the appropriate technical level of all our potential readers presented difficulties; on the other hand, we've tried to build a solid foundation from the ground up and believe that some introductory material benefits everyone. We've also tried to ensure that every reader receives all the fundamental information necessary to truly understand the topics presented.

If you're an experienced Oracle user, you may be tempted to skip over material in this book with which you are already familiar. But experience has shown that some of the most basic Oracle principles can be overlooked, even by experts. We've also seen how the same small "gotchas" trip up even the most experienced Oracle practitioners and cause immense damage if they go unnoticed. After all, an ounce of prevention, tempered by understanding, is worth a pound of cure, especially when you are trying to keep your systems running optimally. So we hope that even experienced Oracle users will find valuable information in every chapter of this book—information that will save hours in their busy professional lives.

Our guiding principle has been to present this information compactly without making it overly tutorial. We think that the most important ratio in a book like this is the amount of useful information you get balanced against the time it takes you to get it. We sincerely hope this volume provides a terrific bang for the buck.

About the Fifth Edition (Oracle Database 12*c*)

The first four editions of this book, covering the Oracle Database up to the Oracle Database 11*g* version, have been well received, and we were pleased that O'Reilly Media agreed to publish this fifth edition. In this update to the book, we have added information describing the latest release of Oracle, Oracle Database 12*c*.

For the most part, the task of preparing this fifth edition was fairly clear-cut, because the Oracle Database 12*c* release is primarily incremental—the new features in the release extend existing features of the database. We've added the information about these extensions to each of the chapters, wherever this information was most relevant and appropriate. However, manageability has greatly changed over the release, and is reflected in many of the most significant changes to content.

Of course, this fifth edition cannot possibly cover everything that is new in Oracle Database 12*c*. In general, we have followed the same guidelines for this edition that we did for the first four editions. If a new feature does not seem to be broadly important, we have not necessarily delved into it. As with earlier editions, we have not tried to produce a laundry list of every characteristic of the Oracle Database. In addition, if a feature falls into an area outside the scope of the earlier editions, we have not attempted to cover it in this edition unless it has assumed new importance.

Structure of This Book

This book is divided into 15 chapters and 2 appendixes, as follows:

Chapter 1 describes the range of Oracle Databases and data stores and Fusion Middleware products and features and provides a brief history of Oracle and relational databases.

Chapter 2 describes the core concepts and structures (e.g., files, processes, pluggable databases, and so on) that are the architectural basis of Oracle.

Chapter 3 briefly describes how to install Oracle and how to configure, start up, and shut down the database and Oracle Net.

Chapter 4 summarizes the various datatypes supported by Oracle and introduces the Oracle objects (e.g., tables, views, indexes). This chapter also covers query optimization.

Chapter 5 provides an overview of managing an Oracle system, including the advisors available as part of Oracle Database 12*c*, the role of Oracle Enterprise Manager (EM) 12*c*, information lifecycle management through the use of heat maps, and working with Oracle Support.

Chapter 6 provides an overview of basic Oracle security, Oracle's security options, basic auditing capabilities, and ways you can leverage database security and audit options to meet compliance needs.

Chapter 7 describes the main issues relevant to Oracle performance—especially the major performance characteristics of disk, memory, and CPU tuning. It describes how Oracle Enterprise Manager, the Automatic Workload Repository, and the Automatic Database Diagnostic Monitor are used for performance monitoring and management, as well as parallelism and memory management in Oracle.

Chapter 8 describes the basic principles of multiuser concurrency (e.g., transactions, locks, integrity problems) and explains how Oracle handles concurrency.

Chapter 9 describes online transaction processing (OLTP) in Oracle.

Chapter 10 describes the basic principles of data warehouses and business intelligence, Oracle Database features used for such solutions, the role of Hadoop in Big Data solu-

tions, Oracle's business intelligence tools, relevant options such as OLAP and data mining, how Oracle's engineered systems fulfill key roles such as in infrastructure, and best practices.

Chapter 11 discusses availability concepts, what happens when the Oracle Database recovers, protecting against system failure, Oracle's backup and recovery facilities, and high availability and failover solutions.

Chapter 12 describes your choice of computer architectures, configuration considerations, and deployment strategies for Oracle, including the array of engineered systems that support that Oracle Database.

Chapter 13 briefly summarizes the Oracle facilities used in distributed processing including two-phase commits and Oracle replication and data transport offerings.

Chapter 14 describes Oracle's object-oriented features, Java's role, Web Services support, multimedia and text extensions to Oracle, XML DB support, spatial capabilities, and the extensibility framework.

Chapter 15 describes cloud definitions, the Oracle Database in the cloud, and the role of APEX.

Appendix A lists the Oracle Database 12*c* changes described in this book.

Appendix B lists a variety of additional resources—both online and offline—so you can do more detailed reading.

Conventions Used in This Book

The following typographical conventions are used in this book:

Italic
: Used for file and directory names, emphasis, and the first occurrence of terms

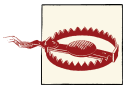
`Constant width`
: Used for code examples and literals

`Constant width italic`
: In code examples, indicates an element (for example, a parameter) that you supply

UPPERCASE
: Generally indicates Oracle keywords

lowercase
: In code examples, generally indicates user-defined items such as variables

This icon indicates a tip, suggestion, or general note. For example, we'll tell you if you need to use a particular version of Oracle or if an operation requires certain privileges.



This icon indicates a warning or caution. For example, we'll tell you if Oracle doesn't behave as you'd expect or if a particular operation negatively impacts performance.

Using Code Examples

This book is here to help you get your job done. Though the nature of this book is such that you will find minimal code, you may use the code in this book in your programs and documentation. You do not need to contact us for permission unless you're reproducing a significant portion of the code. For example, writing a program that uses several chunks of code from this book does not require permission. Selling or distributing a CD-ROM of examples from O'Reilly books *does* require permission. Answering a question by citing this book and quoting example code does not require permission. Incorporating a significant amount of example code from this book into your product's documentation *does* require permission.

We appreciate, but do not require, attribution. An attribution usually includes the title, author, publisher, and ISBN. For example: "*Oracle Essentials: Oracle Database 12c*, Fifth Edition, by Rick Greenwald, Robert Stackowiak, and Jonathan Stern. Copyright 2013 O'Reilly Media Inc., 978-1-4493-4303-3."

If you feel your use of code examples falls outside fair use or the permission given above, feel free to contact us at *permissions@oreilly.com*.

Safari® Books Online

Safari Books Online (*www.safaribooksonline.com*) is an on-demand digital library that delivers expert content in both book and video form from the world's leading authors in technology and business.

Technology professionals, software developers, web designers, and business and creative professionals use Safari Books Online as their primary resource for research, problem solving, learning, and certification training.

Safari Books Online offers a range of product mixes and pricing programs for organizations, government agencies, and individuals. Subscribers have access to thousands of books, training videos, and prepublication manuscripts in one fully searchable database from publishers like O'Reilly Media, Prentice Hall Professional, Addison-Wesley Pro-

fessional, Microsoft Press, Sams, Que, Peachpit Press, Focal Press, Cisco Press, John Wiley & Sons, Syngress, Morgan Kaufmann, IBM Redbooks, Packt, Adobe Press, FT Press, Apress, Manning, New Riders, McGraw-Hill, Jones & Bartlett, Course Technology, and dozens more. For more information about Safari Books Online, please visit us online.

How to Contact Us

Please address comments and questions concerning this book to the publisher:

O'Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472
800-998-9938 (in the United States or Canada)
707-829-0515 (international or local)
707-829-0104 (fax)

We have a web page for this book, where we list errata, examples, and any additional information. You can access this page at *http://bit.ly/oracle-essentials-5th*.

To comment or ask technical questions about this book, send email to *bookquestions@oreilly.com*.

For more information about our books, courses, conferences, and news, see our website at *http://www.oreilly.com*.

Find us on Facebook: *http://facebook.com/oreilly*

Follow us on Twitter: *http://twitter.com/oreillymedia*

Watch us on YouTube: *http://www.youtube.com/oreillymedia*

Acknowledgments

Each of the authors has arrived at this collaboration through a different path, but we would all like to thank the team at O'Reilly for making this book both possible and a joy to write. We'd like to thank our first editor for this edition, Ann Spencer, and the rest of the O'Reilly crew, especially Chris Hearse, the production editor. Also, we'd like to thank our editor from the first three editions, Debby Russell, who was among the first to see the value in such a book and who stepped in to perform final editing on the fifth edition as well. It's incredible how all of these folks were able to strike the perfect balance—always there when we needed something, but leaving us alone when we didn't.

We're all grateful to each other. Giving birth to a book is a difficult process, but it can be harrowing when split multiple ways. Everyone hung in there and did their best throughout this process. We'd also like to give our sincere thanks to the technical re-

viewers for the fifth edition of this book: Penny Avril and Arup Nanda. Thanks as well to reviewers of previous editions that have included Darryl Hurley, Dwayne King, Arup Nanda, Bert Scalzo, Craig Shallahamer of OraPub, Domenick Ficarella, Jonathan Gennick, Jenny Gelhausen, and Dave Klein. This crucially important work really enhanced the value of the book you're reading. And thanks as well to Lance Ashdown for clarifying Oracle Database writes.

Rick thanks the incredibly bright and gifted people who have shared their wealth of knowledge with him over the years, including Bruce Scott, Earl Stahl, Jerry Chang, and many others. In particular, he thanks his first technical mentor, Ed Hickland, who has repeatedly spent time explaining to and discussing with him some of the broader and finer points of database technology.

In subsequent years, Rick has benefitted from a wealth of brilliant co-workers and colleagues, who were always willing to share their views and knowledge, including Graham Wood, Andrew Holdsworth, Tom Kyte, and Bryn Llewellyn. In particular, Rick cherishes both the expertise and friendship of the marvelous Maria Colgan.

For the later editions of this book, Rick would also like to thank all those colleagues at Oracle who helped him in his time of need, checking on those last-minute clarifications, including John Lang, Bruce Lowenthal, Alice Watson, Dave Leroy, Sushil Kumar, Mughees Minhas, Daniela Hansell, and Mark Drake. And a special thank you to Jenny Tsai-Smith, who always seemed to have the time and knowledge to clear up any Oracle Database problem. Rick is part of a fantastic team in development at Oracle, whose members have been a source of advice and friendship. Those members include Mike Hichwa, Patrick Wolf, Jason Straub, Hilary Farrell, Shakeeb Rahman, Colm Divilly, Chris Rice, Joel Kalman, and Dom Lindars. And last, but certainly not least, his primary coauthor, Bob Stackowiak, who has become a good friend over the years of collaboration.

Bob acknowledges all his friends over the years around the world at Oracle Corporation, and from earlier stints at IBM, Harris Computer Systems, and the U.S. Army Corps of Engineers. Through personal relationships and social media, they have shared a lot and provided him with incredible opportunities for learning. At Oracle, he especially thanks members of Andy Mendelsohn's team who have always been helpful in providing material ahead of releases, including George Lumpkin, Hermann Baer, Jean-Pierre Dijcks, Maria Colgan, and many others. Bob and Rick both call out the memory of Mark Townsend for his key role in Oracle's database development over the years and whose talents are missed by all. Bob also extends special thanks to his team in Oracle's Enterprise Solutions Group, especially Alan Manewitz, Louis Nagode, and Art Licht. His management continues to recognize the value of such projects, including David O'Neill and Joe Strada. Paul Cross has served as a mentor over the years. He'd also like to thank his customers, who have always had the most practical experience using the products and tools he has worked with and from whom he continues to learn. Finally, collabo-

rating on books with Rick Greenwald always makes this process fun and has led to other memorable experiences including enjoying live performances of Bruce Springsteen together.

In early editions of this book, Jonathan thanked many of his professional contacts, including Murray Golding, Sam Mele, and the Oracle Server Technologies members and their teams, including Juan Tellez, Ron Weiss, Juan Loaiza, and Carol Colrain for their help during his years at Oracle. And we thank him for all that he gave us in too short a life.

CHAPTER 1
Introducing Oracle

Where do we start? One of the problems in comprehending a massive product such as the Oracle Database is getting a good sense of how the product works without getting lost in the details. This book aims to provide a thorough grounding in the concepts and technologies that form the foundation of Oracle's Database Server, currently known as Oracle Database 12*c*. The book is intended for a wide range of Oracle Database administrators, developers, and users, from the novice to the experienced. It is our hope that once you have this basic understanding of the product, you'll be able to connect the dots when using Oracle's voluminous feature set, documentation, and the many other books and publications that describe the database.

Oracle also offers an Application Server and Fusion Middleware, business intelligence tools, and business applications (the E-Business Suite, PeopleSoft, JD Edwards, Siebel, Hyperion, and Fusion, among others). Since this book is focused on the database, we will only touch on these other software products as they relate to specific Oracle Database topics covered.

This first chapter lays the groundwork for the rest of the book. Of all the chapters, it covers the broadest range of topics. Most of these topics are discussed later in more depth, but some of the basics—for example, the brief history of Oracle and the contents of the different "flavors" of the Oracle Database products—are unique to this chapter.

Over the past 30-plus years, Oracle grew from being one of many vendors that developed and sold a database product to being widely recognized as the database market leader. Although early products were typical of a startup company, the Oracle Database grew such that its technical capabilities are now often viewed as the most advanced in the industry. With each database release, Oracle has improved the scalability, functionality, and manageability of the database.

This book is now in its fifth edition. This edition, like earlier editions, required many changes, since the database has changed a great deal over this time. Highlights of Oracle releases covered in the many editions of this book include:

- Oracle8 (released in 1997) improved the performance and scalability of the database and added the ability to create and store objects in the database.
- Oracle8*i* (released in 1999) added a new twist to the Oracle Database—a combination of enhancements that made the Oracle8*i* Database a focal point in the world of Internet computing.
- Oracle9*i* (released in 2001) introduced Real Application Clusters as a replacement for Oracle Parallel Server and added many management and data warehousing features.
- Oracle Database 10*g* (released in 2003) enabled deployment of "grid" computing. A *grid* is simply a pool of computers and software resources providing resources for applications on an as-needed basis. To support this style of computing, Oracle added the ability to provision CPUs and data. Oracle Database 10*g* also further reduced the time, cost, and complexity of database management through the introduction of self-managing features such as the Automated Database Diagnostic Monitor, Automated Shared Memory Tuning, Automated Storage Management, and Automated Disk Based Backup and Recovery.
- Oracle Database 11*g* (released in 2007) highlighted improvement in self-tuning and managing capabilities, especially in the areas of Automatic Memory Management, partitioning, and security. The lifecycle of database change management was extended within Oracle's Enterprise Manager with improved diagnosis capabilities and linkage to Oracle Support via a Support Workbench. This version also featured improved online patching capabilities. In 2008, Oracle announced that its first engineered system, the Oracle Exadata Database Machine, would support Oracle Database 11*g* Enterprise Edition.
- Oracle Database 12*c* (released in 2013) introduces a number of deployment, manageability, and rapid provisioning features especially useful in private and public *cloud computing* environments where hardware infrastructure and the database are delivered as a service over a network, building upon capabilities introduced in previous releases. Typically, many databases are deployed and managed using this model, so Oracle introduced a capability in the database to share services by defining multitenant container and pluggable databases.

Before we dive into further details, let's step back and look at how databases evolved, how we arrived at the relational model, and Oracle's history. We'll then take an initial look at Oracle Database packaging and key Oracle features today.

The Evolution of the Relational Database

The relational database concept was described first by Dr. Edgar F. Codd in an IBM research publication entitled "System R4 Relational" that was published in 1970. Initially, it was unclear whether any system based on this concept could achieve commercial success. Nevertheless, a company named Software Development Laboratories Relational Software came into being in 1977 and released a product named Oracle V.2 as the world's first commercial relational database within a couple of years (also changing its name to Relational Software, Incorporated). By 1985, Oracle could claim more than 1,000 relational database customer sites. Curiously, IBM would not embrace relational technology in a commercial product until the Query Management Facility in 1983.

Why did relational database technology grow to become the de facto database technology? A look back at previous database technology may help to explain this phenomenon.

Database management systems were first defined in the 1960s to provide a common organizational framework for data formerly stored in independent files. In 1964, Charles Bachman of General Electric proposed a network model with data records linked together, forming intersecting sets of data, as shown on the left in Figure 1-1. This work formed the basis of the CODASYL Data Base Task Group. Meanwhile, the North American Aviation's Space Division and IBM developed a second approach based on a hierarchical model in 1965. In this model, data is represented as tree structures in a hierarchy of records, as shown on the right in Figure 1-1. IBM's product based on this model was brought to market in 1969 as the Information Management System (IMS). As recently as 1980, almost all database implementations used either the network or hierarchical approach. Although several competitors sold similar technologies around 1980, only IMS could still be found in many large organizations just 20 years later.

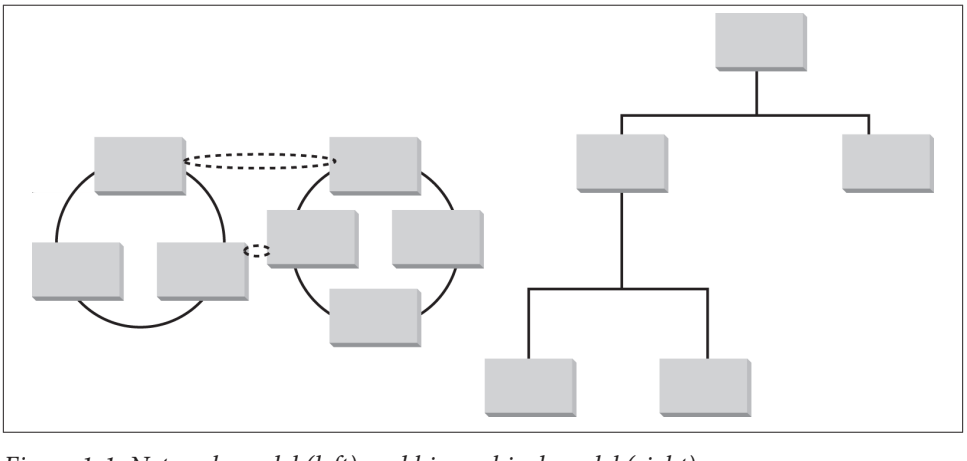
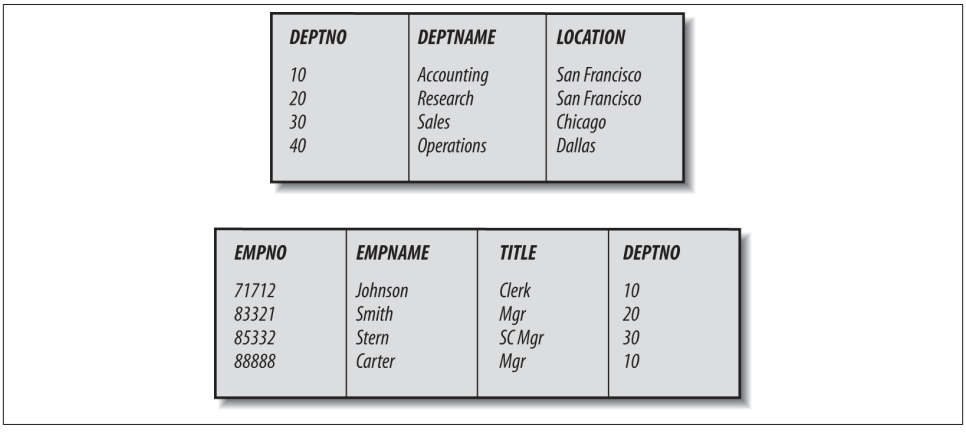


Figure 1-1. Network model (left) and hierarchical model (right)

Relational Basics

The relational database uses the concept of linked two-dimensional tables consisting of rows and columns, as shown in Figure 1-2. Unlike the hierarchical approach, no predetermined relationship exists between distinct tables. This means that data needed to link together the different areas of the network or hierarchical model need not be defined. Because relational users don't need to understand the representation of data in storage to retrieve it (many such business users create ad hoc queries), ease of access combined with improved flexibility to change data models to adapt to changing business needs and helped popularize the relational model.



DEPTNO	DEPTNAME	LOCATION
10	Accounting	San Francisco
20	Research	San Francisco
30	Sales	Chicago
40	Operations	Dallas

EMPNO	EMPNAME	TITLE	DEPTNO
71712	Johnson	Clerk	10
83321	Smith	Mgr	20
85332	Stern	SC Mgr	30
88888	Carter	Mgr	10

Figure 1-2. Relational model with two tables

Relational programming is nonprocedural and operates on a set of rows at a time. In a master-detail relationship between tables, there can be one or many detail rows for each individual master row, yet the statements used to access, insert, or modify the data simply describe the set of results. In many early relational databases, data access required the use of procedural languages that worked one record at a time. Because of this set orientation, programs access more than one record in a relational database more easily. Relational databases can be used more productively to extract value from large groups of data.

The contents of the rows in Figure 1-2 are sometimes referred to as *records*. A column within a row is referred to as a *field*. Tables are stored in a database *schema*, which is a logical organizational unit within the database. Other logical structures in the schema often include the following:

Views
: Provide a single view of data derived from one or more tables or views. The view is an alternative interface to the data, which is stored in the underlying table(s) that makes up the view.

Sequences
: Provide unique numbers, typically used for column values.

Stored procedures
: Contain logical modules that can be called from programs.

Synonyms
: Provide alternative names for database objects.

Indexes
: Provide faster access to table rows.

Database links
: Provide links between distributed databases.

The relationships between columns in different tables are typically described through the use of *keys*, which are implemented through referential integrity constraints and their supporting indexes. For example, in Figure 1-2, you can establish a link between the DEPTNO column in the second table, which is called a *foreign key*, to the DEPTNO column in the first table, which is referred to as the *primary key* of that table.

Finally, even if you define many different indexes for a table, you don't have to understand them or manage the data they contain. Relational databases include a *query optimizer* that chooses whether to use indexes, and the best way to use those indexes, to access the data for any particular query.

The relational approach lent itself to the Structured Query Language (SQL). SQL was initially defined over a period of years by IBM Research, but it was Oracle Corporation that first introduced it to the market in 1979. SQL was noteworthy at the time for being the only language needed for relational databases since you could use SQL:

- For queries (using a SELECT statement)
- As a Data Manipulation Language or DML (using INSERT, UPDATE, and DELETE statements)
- As a Data Definition Language or DDL (using CREATE or DROP statements when adding or deleting tables)
- To set privileges for users or groups (using GRANT or REVOKE statements)

Today, SQL contains many extensions and follows ANSI/ISO standards that define its basic syntax.

How Oracle Grew

In 1983, Relational Software Incorporated was renamed Oracle Corporation to avoid confusion with a competitor named Relational Technologies Incorporated. At this time, the developers made a critical decision to create a portable version of Oracle written in

C (version 3) that ran not only on Digital VAX/VMS systems, but also on Unix and other platforms. By 1985, Oracle claimed the ability to run on more than 30 platforms. Some of these platforms are historical curiosities today, but others remain in use. (In addition to VMS, early operating systems supported by Oracle included IBM MVS, HP/UX, IBM AIX, and Sun's Solaris version of Unix.) Oracle was able to leverage and helped accelerate the growth in popularity of minicomputers and Unix servers in the 1980s. Today, this portability also includes releases for operating systems such as Microsoft Windows and Linux.

In addition to multiple platform support, other core Oracle messages from the mid-1980s still ring true today, including complementary software development and decision support (business intelligence) tools, ANSI standard SQL across platforms, and connectivity over standard networks. Since the mid-1980s, the database deployment model has evolved from single database and application servers to client/server, then to Internet computing implemented using browser-based clients accessing database applications, and now to private and public cloud deployment where the Oracle Database might be deployed as a service (DBaaS) or used as the foundation for a Cloud Platform as a Service (PaaS).

Oracle introduced many innovative technical features to the database as computing and deployment models changed (from offering the first distributed database to supporting the first Java Virtual Machine in the core database engine to enabling grid computing and providing needed services for public and private Cloud deployment). Oracle offered support for emerging standards such as XML, important in deploying a Service-Oriented Architecture (SOA). Table 1-1 presents a short list of Oracle's major product introductions.

Table 1-1. History of Oracle introductions

Year	Feature
1977	Software Development Laboratories founded by Larry Ellison, Bob Miner, Ed Oates
1979	Oracle version 2: first commercially available relational database to use SQL
1983	Oracle version 3: single code base for Oracle across multiple platforms
1984	Oracle version 4: with portable toolset, read consistency
1986	Oracle version 5 generally available: client/server Oracle relational database
1987	CASE and 4GL toolset
1988	Oracle Financial Applications built on relational database
1989	Oracle6 generally available: row-level locking and hot backups
1991	Oracle Parallel Server on massively parallel platforms
1993	Oracle7: with cost-based optimizer
1994	Oracle version 7.1 generally available: parallel operations including query, load, and create index
1996	Universal database with extended SQL via cartridges, thin client, and application server
1997	Oracle8 generally available: object-relational and Very Large Database (VLDB) features

Year	Feature
1999	Oracle8*i* generally available: Java Virtual Machine (JVM) in the database
2000	Oracle9*i* Application server generally available: Oracle tools integrated in middle tier
2001	Oracle9*i* Database Server generally available: Real Application Clusters, OLAP, and data mining in the database
2003	Oracle Database 10*g* and Oracle Application Server 10*g*: "grid" computing enabled; Oracle Database 10*g* automates key management tasks
2005	Oracle completes PeopleSoft acquisition and announces Siebel acquisition, thus growing ERP and CRM applications and business intelligence offerings
2007	Oracle Database 11*g*: extension of self-managing capabilities and end-to-end database change management; Hyperion acquisition adds database-independent OLAP and Financial Performance Management applications; Oracle Virtual Machine (Oracle VM) announced
2008	Oracle acquires BEA Systems (middleware software); Oracle's first engineered system, Oracle Exadata, is introduced for data warehousing
2009	Oracle Exadata featuring Smart Flash Cache is enhanced as a platform for all Oracle Database use cases, including data warehousing, transaction processing, and database consolidation
2010	Oracle completes Sun acquisition; Oracle Exalogic Elastic Cloud engineered system introduced
2011	Oracle Database Appliance, SuperCluster, Exalytics (for business intelligence), and Big Data Appliance are introduced; Fusion Applications announced as available
2012	Oracle announces Oracle Database 12*c*: support for public and private Cloud deployment with multitenant container databases, pluggable databases, and improved management capabilities highlighted; Oracle continues acquisitions of Cloud-based applications solutions
2013	Oracle Database 12*c* generally available

The Oracle Database Family

Oracle Database 12*c* is the most recent version of the Oracle Relational Database Management System (RDBMS) family of products that share common source code. The family of database products includes:

Oracle Enterprise Edition
: Flagship database product and main topic of this book, aimed at large-scale implementations that require Oracle's full suite of database features and options. For advanced security, only the Enterprise Edition features Virtual Private Database (VPD) support, Fine-Grained Auditing, and options including the Database Vault, Advanced Security, and Label Security. Data warehousing features only in Enterprise Edition include compression of repeating stored data values, cross-platform transportable tablespaces, Information Lifecycle Management (ILM), materialized views query rewrite, and the Partitioning, OLAP, and Advanced Analytics Options. High-availability features unique to the Enterprise Edition include Data Guard and Flashback Database, Flashback Table, and Flashback Transaction Query. The Enterprise Edition is the database version supported on Oracle's engineered systems.

Oracle Standard Edition
: Oracle's database intended for small- and medium-sized implementations. This database can be deployed onto server configurations containing up to 4 CPUs on a single system or on a cluster using Real Application Clusters (RAC).

Oracle Standard Edition One
: Designed for small implementations, this database can support up to 2 CPUs and does not support RAC. The feature list is otherwise similar to Oracle Standard Edition.

Oracle Personal Edition
: Database used by single developers to develop code for implementation on Oracle multiuser databases. It requires a license, unlike Express Edition, but gives you the full Enterprise Edition set of functionality.

Oracle Express Edition
: Entry-level database from Oracle available at no charge for Windows and Linux and unsupported as a product, this database is limited to 1 GB of memory and 4 GB of disk. It provides a subset of the functionality in Standard Edition One, lacking features such as a Java Virtual Machine, server-managed backup and recovery, and Automatic Storage Management. Although this database is not manageable by Oracle Enterprise Manager, you can deploy it for and manage multiple users through the Oracle Application Express administration interface.

Oracle releases new versions of the flagship database every three to five years. New releases typically follow themes and introduce a significant number of new features. In recent releases, these themes are indicated in the product version naming. In 1998, Oracle announced Oracle8*i*, with the "*i*" added to denote new functionality supporting Internet deployment. Oracle9*i* continued using this theme. In 2003, Oracle announced Oracle Database 10*g*, with the "*g*" denoting Oracle's focus on emerging grid computing deployment models, then announced Database 11*g* with further improvements in manageability in 2007. In 2012, Oracle announced Oracle Database 12*c*, the "*c*" denoting new database functionality supporting Cloud deployment. In between major versions, Oracle issues point releases that also add features but are more typically focused on improvements to earlier capabilities.

The terms "Oracle," "Oracle Database," "database," "Oracle8," "Oracle8*i*," "Oracle9*i*," "Oracle Database 10*g*," "Oracle Database 11*g*," and "Oracle Database 12*c*" might appear to be used somewhat interchangeably in this book because Oracle Database 12*c* includes all the features of previous versions. When we describe a new feature that was first made available specifically in a certain release, we've tried to note that fact to avoid confusion, recognizing that many of you maintain older releases of Oracle. We typically use the simple terms "Oracle" and "database" when describing features that are common to all these releases.

Oracle Development has developed releases using a single source code model for the core family of database products since 1983. While each database implementation includes some operating-system-specific source code at very low levels in order to better leverage specific platforms, the interfaces that users, developers, and administrators deal with for each version are consistent. This development strategy enables Oracle to focus on implementing new features only once across its product set.

The introduction of Oracle's engineered systems, Exadata storage, and the Exadata Storage Server software enabled Oracle to optimize the database for specific hardware server and storage configurations. Today, Oracle offers a family of engineered systems capable of running the Oracle Database. The Oracle Exadata Database Machine was the most popular engineered system as this edition of the book was published. The Oracle SuperCluster is a general purpose platform also featuring Exadata storage (and some general purpose storage) and is designed to run both the Oracle Database and Oracle Fusion Middleware. The Oracle Database Appliance is a two-node configuration designed to be a smaller departmental server. The Exalogic Elastic Cloud system is optimally designed to run Oracle's Fusion Middleware and often is deployed as a middle-tier server in front of Exadata, but occasionally is deployed with the Oracle Database also running on it. Neither the Oracle Database Appliance nor Exalogic Elastic Cloud support Exadata storage.

Summary of Oracle Database Features

The Oracle Database is a broad and deep product. To give some initial perspective, we begin describing Oracle with a high-level overview of the basic areas of functionality. By the end of this portion of the chapter, you will have orientation points to guide you in exploring the topics in the rest of this book.

To give some structure to the broad spectrum of the Oracle Database, we've organized our initial discussion of these features and complementary software components into the following sections:

- Database application development features
- Database connection features
- The role of Oracle Fusion Middleware
- Distributed database features
- Data movement features
- Database performance features
- Managing the Oracle Database
- Database security features

- Database development tools

In this chapter, we've included a lot of terminology and rather abbreviated descriptions of features. Oracle is a huge system. Our goal here is to quickly familiarize you with the full range of features in the system and introduce the concepts we are covering in this book. Subsequent chapters will provide additional details about these features and concepts. Obviously, though, whole books have been written about some of the feature areas summarized here, so this book is often used as a starting point in understanding where to go next.

Database Application Development Features

The Oracle Database is typically used to store and retrieve data through applications. The features of the Oracle Database and related products described in this section are used to create applications. We've divided the discussion in the following subsections into database programming and database extensibility options. Later in this chapter, we will describe Oracle's development tools and Oracle's other embedded database products that meet unique applications deployment needs.

Database Programming

All flavors of the Oracle Database include languages and interfaces that enable programmers to access and manipulate the data in the database. Database programming features usually interest developers who are creating Oracle-based applications to be sold commercially, or IT organizations building applications unique to their businesses. Data in Oracle can be accessed using SQL, SQL/XML, XQuery, and WebDAV. Programs deployed within the database can be written in PL/SQL and Java.

SQL

The ANSI standard Structured Query Language (SQL) provides basic functions for data manipulation, transaction control, and record retrieval from the database. Most business users of the database interact with Oracle through applications or business intelligence tools that provide interfaces hiding the underlying SQL and its complexity.

PL/SQL

Oracle's PL/SQL, a procedural language extension to SQL, is commonly used to implement program logic modules for applications. PL/SQL can be used to build stored procedures and triggers, looping controls, conditional statements, and error handling. You can compile and store PL/SQL procedures in the database. You can also execute PL/SQL blocks via SQL*Plus, an interactive tool provided with all versions of Oracle. PL/SQL program units can be precompiled. Additionally, Oracle supplies a lot of ad-

ditional functionality using PL/SQL programs included with the database, referred to as packages, which can be called from standard PL/SQL code.

Java

Oracle8*i* introduced the use of Java as a procedural language and a Java Virtual Machine (JVM) in the database (originally called JServer). The JVM includes support for Java stored procedures, methods, triggers, Enterprise Java Beans (EJBs), CORBA, IIOP, and HTTP.

The inclusion of Java within the Oracle Database allows Java developers to leverage their skills as Oracle applications developers. Java applications can be deployed in the client, Application Server, or database, depending on what is most appropriate. Current Oracle Database versions include a just-in-time Java compiler that is enabled by default. The importance of Java to Oracle is illustrated by the acquisition of Sun by Oracle in 2010 and continued development efforts around Java since.

Oracle and Web Services

As of Oracle Database 11*g*, the Database can serve as a Web Services provider implemented through XML DB in the database. Web services enable SQL or XQuery to submit queries and receive results as XML, or invoke PL/SQL functions or package functions and to receive results. XQuery provides support for the JSR standard and is further optimized for performance in the Oracle Database. As of Oracle Database 12*c*, XQuery updates are supported. You can also define RESTful Web Services to access both SQL and PL/SQL in an Oracle Database through the APEX Listener, described in Chapter 15.

Large objects

The Oracle Database has been able to store large objects since Oracle8 added the capability to store multiple LOB columns in each table. Oracle Database 10*g* essentially removed the space limitation on large objects. Oracle Database 11*g* greatly improved the performance of query and insert operations when used with LOBs through the introduction of SecureFiles. SecureFiles serve as a place to securely store LOBs in the Oracle Database instead of in filesystems while delivering performance similar to that experienced when LOBs are stored in filesystems. Transparent Data Encryption, a security feature described below and later in the book, is supported for SecureFiles LOB data.

Object-oriented programming

Support of object structures has existed since Oracle8*i* to provide support for an object-oriented approach to programming. For example, programmers can create user-defined data types, complete with their own methods and attributes. Oracle's object support includes a feature called Object Views through which object-oriented programs can make use of relational data already stored in the database. You can also store objects in the database as varying arrays (VARRAYs), nested tables, or index-organized tables (IOTs).

Third-generation languages (3GLs)

Programmers can interact with the Oracle Database from C, C++, Java, or COBOL by embedding SQL in those applications. Prior to compiling the applications using a platform's native compilers, you must run the embedded SQL code through a precompiler. The precompiler replaces SQL statements with library calls the native compiler can accept. Oracle provides support for this capability through optional "programmer" precompilers for C and C++ using Pro*C and for COBOL using Pro*COBOL. In recent Oracle versions, Oracle features SQLJ, a precompiler for Java that replaces SQL statements embedded in Java with calls to a SQLJ runtime library, also written in Java.

Database drivers

All versions of Oracle include database drivers that allow applications to access Oracle via ODBC (the Open Database Connectivity standard) or JDBC (the Java Database Connectivity open standard). Also available are Oracle Data Access Connectors (ODAC) for .NET. ODAC provides a data provider for .NET, providers for ASP.NET and .NET stored procedures, and tools for developers using Visual Studio.

The Oracle Call Interface

If you're an experienced programmer seeking optimum performance or a finer level of control, you may choose to define SQL statements within host-language character strings and then explicitly parse the statements, bind variables for them, and execute them using the Oracle Call Interface (OCI). OCI is a much more detailed interface that requires more programmer time and effort to create and debug. Developing an application that uses OCI can be time-consuming, but the added functionality and incremental performance gains could make spending the extra time worthwhile.

In certain programming scenarios, OCI improves application performance or adds functionality. For instance, in high-availability implementations in which multiple systems share disks using Real Application Clusters, you could write programs using OCI that allow users to reattach to a second server transparently if the first fails. As of Oracle Database 12*c*, the Transaction Guard API to the database can be used in order to preserve guaranteed commits where data is accessed via OCI (or alternatively via JDBC thin drivers, OOCI, or ODP.NET).

National Language Support

National Language Support (NLS) provides character sets and associated functionality, such as date and numeric formats, for a variety of languages. The initial release of Oracle Database 12*c* features Unicode 6.1 support. All data may be stored as Unicode, or select columns may be incrementally stored as Unicode. UTF-8 encoding and UTF-16 encoding provide support for more than 57 languages and 200 character sets. Extensive localization is provided (for example, for data formats), and customized localization can be added through the Oracle Locale Builder. Oracle includes a Globalization Toolkit for creating applications that will be used in multiple languages.

Database Extensibility

The Internet and corporate intranets have created a growing demand for storage and manipulation of nontraditional data types within the database. There is a need for extensions to the standard functionality of a database for storing and manipulating image, audio, video, spatial, and time series information. These capabilities are enabled through extensions to standard SQL.

Oracle Multimedia

Oracle Multimedia (formerly *interMedia*) provides text manipulation and additional image, audio, video, and locator functions in the database. Oracle Multimedia offers the following major capabilities:

- The text portion of Multimedia (Oracle Text) can identify the gist of a document by searching for themes and key phrases within the document.
- The image portion of Multimedia can store and retrieve images of various formats; since Oracle Database 11*g*, DICOM medical images are supported in the database.
- The audio and video portions of Multimedia can store and retrieve audio and video clips, respectively.
- The locator portion of Multimedia can retrieve data that includes spatial coordinate information.

Oracle Spatial and Graph Option

The Spatial and Graph Option is available for the Oracle Enterprise Edition. This option can be used to optimize the display and retrieval of data linked to coordinates, determine distance, and compute other geometric values such as area. It is often is used in the development of spatial information systems by vendors of Geographic Information Systems (GIS) products. Oracle Database 12*c* added support of named graphs in the database as defined by the Worldwide Web Consortium (W3C) in its Resource Description Framework (RDF).

XML DB

Oracle added native XML data type support to the Oracle9*i* Database with XML and SQL interchangeability for searching. The structured XML object is held natively in object relational storage, meeting the W3C DOM specification. XML standards supported include XML Schema, XPath (syntax for searching in SQL), XQuery, XSLT, and DOM. XMLIndex can be used with all forms of XML data in the database. As of Oracle Database 12*c*, XML DB is a mandatory part of the database and cannot be uninstalled.

Database Connection Features

The connection between the client and the database server is a key component of the overall architecture. The database connection is responsible for supporting all communications between an application and the data it uses. Database users connect to the database by establishing a network connection. You can also link database servers via network connections. Oracle provides a number of features to establish connections between users and the database and/or between database servers, as described in the following subsections.

Oracle Net Services

Oracle's Net Services provide the interface between networks and distributed Oracle Databases establishing database sessions for purposes of distributed processing. You can use Oracle Net Services over a wide variety of network protocols including TCP/IP, HTTP, FTP, and WebDAV. The Services include Oracle Net, used to establish sessions, and the Oracle Database Server Listener. Client requests can be handled through dedicated or shared servers.

Oracle Internet Directory

The Oracle Internet Directory (OID) is an LDAP (Lightweight Directory Access Protocol) directory and supports Oracle Net and other LDAP-enabled protocols. Database support first appeared in Oracle8*i* and replaced Oracle Names, which was used to enable user connections to an Oracle Database server without having a client-side configuration file. The directory services are provided by the Oracle Fusion Middleware Identity Management platform.

Oracle Connection Manager

Each connection to the database takes up valuable network resources, which can impact the overall performance of a database application. Oracle's Connection Manager (CMAN), illustrated in Figure 1-3, reduces the number of Oracle Net client network connections to the database through the use of *concentrators*, which provide connection multiplexing to implement multiple connections over a single network connection. Connection multiplexing provides the greatest benefit when there are a large number of active users.

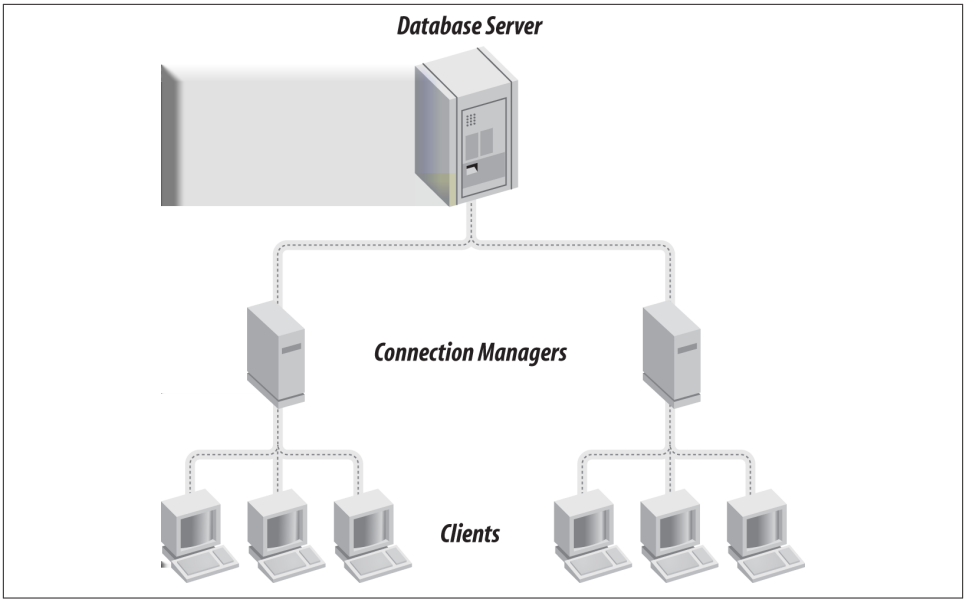


Figure 1-3. Concentrators with Connection Managers for a large number of users

You can also use the Connection Manager to provide multiprotocol connectivity if you still have some clients and servers not using TCP/IP. Oracle Database 10*g* first introduced the dynamic Connection Manager configuration, enabling the changing of CMAN parameters without shutting down the CMAN process.

The Role of Oracle Fusion Middleware

The growing popularity of Internet and intranet applications in the late 1990s led to a change in deployment from client/server (with fat clients running a significant piece of the application) to a three-tier architecture (with a browser supplying everything needed for a thin client). Hence, middleware is an important part of any database connection strategy discussion. Oracle's WebLogic Server enables deployment of the middle tier in a three-tier solution for web-based applications, component-based applications, and enterprise application integration. Oracle WebLogic Server is a key part of Oracle's Fusion Middleware, replacing Oracle's earlier generation Internet Application Server, and is a component of Oracle's Cloud Application Foundation.

Other Fusion Middleware components address transaction management, data integration, business process management, business intelligence, identity management, service-oriented architectures (SOA), and portal, social, and content platforms (WebCenter). We'll introduce those after first describing the WebLogic Server.

Oracle's WebLogic Server

The Oracle WebLogic Server comes in two editions, a Standard Edition and an Enterprise Edition. Both editions include support for the latest Java Enterprise Edition (EE) specification. At the time this edition of the book was published, the WebLogic Server was fully Java EE 6 compatible with the following: JSF 2.0, Servlet 3.0, EJB 3.1, Java Persistence API (JPA) 2.0, CDI 1.0, JAX-RS 1.1, and Java Bean Validation 1.0. Reference implementations used include the EclipseLink JPA and Jersey 1.1. (Oracle also offers an open source application server named Oracle GlassFish that is based on the Java EE 6 specification but not based upon the WebLogic Server—it is often positioned by Oracle as more of a lightweight platform and is less likely to appear where Oracle Databases are deployed as enterprise class solutions.)

The WebLogic Server Standard Edition includes the Hotspot and JRockit Java Virtual Machines (JVMs). Supported development and testing environments include:

TopLink
: TopLink provides a Java persistence framework that includes support for Object-Relation Mapping with the JPA, Object-XML Binding with JAXB and SDO, and Database Web Services for data access using JAX-WS. It can be used to link Java objects to the Oracle Database via JDBC such that the Java developer need not build SQL calls or face broken Java applications resulting from database schema changes.

Application Development Framework (ADF)
: ADF is built upon the Enterprise Java platform and provides infrastructure and development framework for Model View Controller (MVC) layers of an application.

JDeveloper
: JDeveloper is Oracle's Java Integrated Development Environment (IDE) for the Oracle Database and Fusion Middleware.

Classloader Analysis Tool (CAT)
: CAT in the WebLogic Server is used for finding Java class and library conflicts.

Other development environments
: Oracle provides Eclipse plug-ins and support for the NetBeans IDE, both of which are open source Java EE development offerings. WebLogic Server also provides a Maven plug-in for testing and development and can support and leverage Spring applications.

For deployment, the WebLogic Server Standard Edition features an administration console with a change center, a WebLogic Diagnostic Framework (WLDF), and support for command line and scripting control.

The Oracle WebLogic Server Enterprise Edition adds the following capabilities:

Clustering and Failover
: WebLogic Servers clusters are load balanced and self-monitored to avoid overloading. If problems occur, servers can be migrated in whole, service can be automatically migrated, and the transaction recovery service can be invoked.

High performance and reliable Java Message Service (JMS)
: JMS improvements include support for automatic migration of services from a failing server to a healthy one, store and forward messaging, and support for Oracle Database Streams Advanced Queuing (AQ).

Oracle Virtual Assembly Builder (VAB)
: VAB enables packaging of software components for movement among development, test, and production environments and can be used with Oracle VM to create, configure, and provision applications to virtualized resources.

JRockit Mission Control
: JRockit Mission Control provides JVM diagnostics using an Eclipse-based user interface and includes JRockit Flight Recorder (JFR) for further analysis of problems after they occur.

Oracle Traffic Director
: The Oracle Traffic Director optimizes performance of the WebLogic Server Enterprise Edition when deployed on the Exalogic Elastic Cloud engineered system from Oracle.

Oracle also offers a WebLogic Suite designed for highly available and scalable Cloud deployment and includes the WebLogic Server Enterprise Edition and Coherence Enterprise Edition in this packaging. Coherence enables the pooling and sharing of memory across multiple servers and is managed using the WebLogic administration and scripting tools. TopLink Grid is used when deploying JPA applications in such deployments.

Other suites that are options for the WebLogic Suite include:

Business Process Management (BPM) Suite
: The Business Process Management Suite includes BPM Studio client modeling, BPM (process) Composer, BPMN Service Engine and Workflow Extensions (for BPMN 2.0 and BPEL, the Business Process Engineering Language), BPM Process Spaces, and BPM Process Analytics (integrated with Oracle Business Intelligence and Oracle Business Intelligence Strategy Maps and Balanced Scorecards).

SOA Suite for Oracle Middleware
: This suite bundles Oracle Fusion Middleware SOA offerings, including BPEL Process Manager, Business Activity Monitoring (BAM) for real-time alerting and dashboards, business rules engine, Enterprise Service Bus (for messaging, routing,

and transformations), the Aqualogic Service Bus, Web Services Management (including a policy manager and monitoring dashboard), Web Services Registry, applications and technology adapters, and Oracle Event Processing.

The Fusion Middleware SOA Suite serves as the basis for Oracle's Application Integration Architecture (AIA). AIA also includes prepackaged business objects and business processes known as Process Integration Packs and provides key underpinnings used in integrating Oracle's applications.

Oracle Tuxedo

One could argue that the concept of a middle-tier platform began prior to the popularity of the Internet because of a need for transaction monitors. A transaction monitor is used to ensure data integrity across all resources (even if resource failures occur), enables distributed applications to work together seamlessly, and enables publish and subscribe event models. Oracle Tuxedo has a long history of providing these capabilities that evolved under the ownership of several vendors.

Oracle Tuxedo supports the ATMI programming model, the X/Open API for C, C++, COBOL, and Java. It also supports the SCA and CORBA programming models. A variety of LDAP servers are supported for authentication and authorization. An IDE is provided that supports metadata-driven development.

Tuxedo supervises two-phase commits for transactions via the XA protocol. It provides its own message queuing capability and has bi-directional adapters for IBM CICS/IMS applications and access to IBM WebSphere MQ queues. It is monitored through the Oracle Tuxedo System and Application Monitor (TSAM) and is integrated into Enterprise Manager.

Data Integration Tools

Oracle's data integration products that are considered to be part of Fusion Middleware include the following:

Oracle Data Integrator (ODI)
: ODI is an extraction, transformation, and loading declarative design tool sometimes referenced as an ELT (extraction, load, and transformation) tool by Oracle as transformations are pushed into the target database. Knowledge Modules define the technical implementation of the integration process. A variety of source and target databases are supported. ODI has replaced Oracle Warehouse Builder as Oracle's primary offering for ELT.

Enterprise Data Quality for Oracle Data Integrator
: Enterprise Data Quality Options for ODI are available for data profiling, batch processing, and address verification.

Oracle GoldenGate

GoldenGate enables lightweight capture, routing, transformation, and delivery of changed data among a variety of Oracle and non-Oracle databases in near real time. The Capture software component resides on the source database or monitors JMS messaging for changes. Trail Files containing changed data reside on the source or target server. The Delivery Module takes changed data from the Trail Files and applies it to the target database.

Business Intelligence Tools

Oracle data warehouses are often accessed using business intelligence tools from a variety of vendors. Usage of Oracle's own tools became more common for such deployment as Oracle grew its offerings through acquisitions, replacing Oracle's earlier Discoverer and Reports products.

Oracle's flagship product in this area is Oracle Business Intelligence Foundation Suite, consisting of former Siebel Analytics/Oracle Business Intelligence Enterprise Edition and Hyperion Essbase components. The product has evolved into an integrated middle-tier platform that features reporting, ad hoc query and analysis, dashboards, balanced scorecard and strategy management, Office plug-ins, and mobile support.

Oracle also offers business intelligence applications that include data models and reporting and analysis with pre-populated business metadata. Applications include Oracle's Business Intelligence Applications (the former Siebel Business Analytics applications) and Hyperion Financial Performance Management applications.

Other business intelligence offerings include Endeca Information Discovery (EID) and Real-Time Decisions (RTD). EID features a multifaceted server for data discovery that provides multiple drill paths through structured and unstructured data without the need to pre-define a schema. RTD is a real-time recommendation engine that can mine data or use input from other data mining and statistics models in order to present best recommendations to web interfaces and through applications.

WebCenter

The WebCenter Suite of products features WebCenter Portal and WebCenter Content. WebCenter Portal is Oracle's enterprise portal framework based on the Oracle ADF and used for creating dynamic portals, websites, composite applications, and mash-ups. Social collaboration is enabled through services such as discussion forums, wikis, blogs, RSS, tags, links, social networking, and activity streams.

WebCenter Content is a content management platform that provides version control, security, indexing and search, metadata, a workflow engine, and replication. Capabilities include content publishing, digital asset management, document imaging, records and

retention management, and archiving and storage management through the WebCenter Portal.

Identity Management

Oracle's Identity Management platform for Fusion Middleware includes Oracle's Access Management Suite, Oracle Identity Governance, and Oracle Directory Services. This suite of products enables the securing of applications and associated data, Web Services, and Cloud-based services.

The Access Management Suite provides authentication, single sign-on, mobile and social sign-on, entitlement management, and fine-grained authentication. Key components of the Access Management Suite include Access Manager, Adaptive Access Manager, Identity Federation, Entitlements Server, OpenSSO Fedlet, and Security Token Service (STS). External authorization is available through support of multiple standards including XACML, NIST, and Enterprise RBAC.

The Identity Governance Suite provides a platform for access requests, role lifecycle management, access certification, closed loop remediation, and privileged account management. A lengthy list of components are included: Identity Analytics, Identity Manager, Privileged Account Manager, Identity Manager Connector for Database User Management, Identity Manager Connector for Microsoft Active Directory, Identity Manager Connector for Microsoft Windows, Identity Manager Connector for Novell eDirectory, Identity Manager Connector for Oracle Internet Directory, Identity Manager Connector for Sun Java System Directory, and Identity Manager Connector for Unix.

Oracle's Directory Services include the Virtual Directory, Oracle Internet Directory including Delegated Administration Service (DAS) and Directory Integration Platform (DIP), Directory Server Enterprise Edition, and Oracle Unified Directory. The Unified Directory is a Java-based directory service adhering to LDAP and Directory Services Markup Language standards with advanced storage, proxy, synchronization, and virtualization capabilities.

Distributed Database Features

The Oracle Database is well known for its ability to handle extremely large volumes of data and users. Oracle not only scales through deployment on increasingly powerful single platforms, but it also can be deployed in distributed configurations. Oracle deployed on multiple platforms can be combined to act as a single logical distributed database.

This section describes some of the basic ways that Oracle handles database interactions in a distributed database system.

Distributed Queries and Transactions

Data within an organization is often spread among multiple databases for reasons of both capacity and organizational responsibility. Users may want to query this distributed data or update it as if it existed within a single database.

Oracle first introduced distributed databases in response to the requirements for accessing data on multiple platforms in the early 1980s. *Distributed queries* can retrieve data from multiple databases. *Distributed transactions* can insert, update, or delete data on distributed databases. Oracle's two-phase commit mechanism guarantees that all the database servers that are part of a transaction will either commit or roll back the transaction. Background recovery processes can ensure database consistency in the event of system interruption during distributed transactions. Once the failed system comes back online, the same process will complete the distributed transactions.

Distributed transactions can also be implemented using popular transaction monitors (TPs) such as Tuxedo that interact with Oracle via XA, an industry standard (X/Open) interface.

Heterogeneous Services

Heterogeneous Services allow non-Oracle data and services to be accessed from an Oracle Database through generic connectivity via ODBC and OLE-DB, which are included with the database.

Optional Transparent Gateways use agents specifically tailored for a variety of target systems. Transparent Gateways allow users to submit Oracle SQL statements to a non-Oracle distributed database source and have them automatically translated into the SQL dialect of the non-Oracle source system, which remains transparent to the user. Gateways are available for Sybase, Microsoft SQL Server, DRDA, Informix, Teradata, APPC, and WebSphere MQ.

In addition to providing underlying SQL services, Heterogeneous Services provide transaction services utilizing Oracle's two-phase commit with non-Oracle databases and procedural services that call third-generation language routines on non-Oracle systems. Users interact with the Oracle Database as if all objects are stored in the Oracle Database, and Heterogeneous Services handle the transparent interaction with the foreign database on the user's behalf. Oracle 12c includes a new feature called SQL Translation, which allows the Oracle Database to dynamically translate SQL from a different database. This feature makes it easier to migrate data to an Oracle Database without the need for excessive rewrites of SQL in your applications. The feature is described in Chapter 4.

Data Movement Features

Moving data from one Oracle Database to another is often a requirement when using distributed databases, or when a user wants to implement multiple copies of the same database in multiple locations to reduce network traffic or increase data availability. You can export data and data dictionaries (metadata) from one database and import them into another. Oracle Database 10*g* introduced a high-speed data pump for the import and export.

Oracle also offers many other advanced features in this category, including transportable tablespaces and Advanced Queuing—/Oracle Streams. We introduce these in the next section. Pluggable databases are a key new feature in Oracle Database 12*c* that can be used to move data from one instance to another. We'll describe the impact of pluggable databases in Oracle Database 12*c* in the manageability section of this chapter.

Transportable Tablespaces

Transportable tablespaces first appeared in Oracle8*i*. Instead of using the export/import process, which dumps data and the structures that contain it into an intermediate file for loading, you can place a tablespace in read-only mode, move or copy it from one database to another, and then mount it. The same data dictionary information (metadata) describing the tablespace must exist on the source and the target. This feature can save a lot of time since it simplifies the movement of large amounts of data. Starting with Oracle Database 10*g*, you can move data with transportable tablespaces between heterogeneous platforms or operating systems.

Advanced Queuing and Oracle Streams

Advanced Queuing (AQ) was introduced in Oracle8 to provide a means to asynchronously send messages from one Oracle Database to another. Messages are stored in a queue in the database and sent asynchronously when a connection is made, so the amount of overhead and network traffic is much lower than it would be using traditional guaranteed delivery through the two-phase commit protocol between source and target. This approach enabled a *content-based publish and subscribe solution* using a rules engine to determine relevant subscribing applications. In Oracle9*i*, AQ added XML support and Oracle Internet Directory (OID) integration.

AQ became part of Oracle Streams in the second release of Oracle9*i* and features log-based replication for data capture, queuing for data staging, and user-defined rules for data consumption. However, Oracle now recommends GoldenGate in its Fusion Middleware for enabling change data capture and replication as it provides more flexibility since it supports Oracle and a variety of non-Oracle databases.

Database Performance Features

Oracle includes many features specifically designed to boost performance in certain situations. We've divided the discussion in the following subsections into two categories: database parallelization and data warehousing.

Database Parallelization

By breaking up a single task into smaller tasks and assigning each subtask to an independent process, you can dramatically improve the performance of certain types of database operations. Database tasks that are implemented in parallel speed the querying, tuning, and maintenance of the database. Examples of query features implemented in parallel in Oracle Database 12*c* include:

- Table scans
- Nested loops
- Sort merge joins
- GROUP BYs
- NOT IN subqueries (anti-joins)
- User-defined functions
- Index scans
- Select distinct UNION and UNION ALL
- Hash joins
- ORDER BY and aggregation
- Bitmap star joins
- Partition-wise joins
- Concurrent Union-All
- Correlated filters and expressions
- Stored procedures (PL/SQL, Java, external routines)

In addition to parallel query, many other Oracle features and capabilities can be parallelized.

Data Warehousing

While parallel features improve the overall performance of the Oracle Database, the Oracle Database also has particular performance enhancements for business intelligence and data warehousing applications that we briefly introduce in this section. Of course, as the Oracle BI Server most often accesses relational databases, these features

can greatly speed the performance of that tool and other non-Oracle business intelligence tools.

Bitmap indexes

Stored bitmap indexes have been available in the Oracle Database since Oracle 7.3 and provide a fast way of selecting and retrieving certain types of data. Bitmap indexes typically work best for columns that have few different values relative to the overall number of rows in a table.

Rather than storing the actual value, a bitmap index uses an individual bit for each potential value with the bit either "on" (set to 1) to indicate that the row contains the value or "off" (set to 0) to indicate that the row does not contain the value.

Star query optimization

Typical data warehousing queries occur against a large *fact table* with foreign keys to much smaller *dimension tables*. Oracle added an optimization for queries against this type of *star schema* in Oracle 7.3. Performance gains are realized through the use of Cartesian product joins of dimension tables with a single join back to the large fact table. Oracle8 introduced a mechanism called a *parallel bitmap star join*, which uses bitmap indexes on the foreign keys to the dimension tables to speed star joins involving a large number of dimension tables.

Materialized views

Since Oracle8*i*, materialized views have provided another means of achieving a significant speedup of query performance. Summary-level information derived from a fact table and grouped along dimension values is stored as a materialized view. Queries that can use this view are directed to the view, transparently to the user and the SQL they submit. Oracle has continued to improve optimizer usage of materialized views with each new release of the database.

Analytic functions

A growing trend in Oracle and other databases is inclusion of SQL-accessible analytic and statistical functions in the database. Oracle first introduced such capabilities in Oracle8*i* with the CUBE and ROLLUP functions. Today, the functionality provided also includes ranking functions, windowing aggregate functions, lag and lead functions, reporting aggregate functions, statistical aggregates, linear regression, descriptive statistics, correlations, crosstabs, hypothesis testing, distribution fitting, and Pareto analysis.

OLAP Option

The OLAP Option physically stores dimensionally aware cubes in the Oracle relational database. These cubes are most frequently accessed using SQL, although a Java API is also supported. Since Oracle Database 11*g*, the Oracle Database optimizer recognizes the levels within these cubes. As a result, any business intelligence tool that submits SQL

to an Oracle Database can transparently take advantage of the improved performance offered by deployment of this option. Refreshes of the values in these cubes are now maintained similar to refreshing materialized views.

Advanced Analytics Option

Since Oracle9*i*, popular data mining algorithms have been embedded in the database through the Data Mining Option and are exposed through a PL/SQL or Java data mining API. Data mining applications that use these algorithms are typically built using Oracle's DataMiner. Data mining algorithms available for Oracle Database 12*c* include Naïve Bayes, Associations, Adaptive Bayes Networks, Clustering, Support Vector Machines (SVM), Nonnegative Matrix Factorization (NMF), Decision Trees, and Generalized Linear Models.

Oracle R Enterprise support was added to the Oracle Data Mining Option that was then renamed the Oracle Advanced Analytics Option in later releases of Oracle Database 11*g*. This capability enables R statistical programmers using open source tools to generate R scripts and then deploy those scripts in the Oracle Database. This eliminates the need to move data out of the Oracle Database onto a separate analysis platform and scales consistently with the Oracle Database.

Managing the Oracle Database

Oracle includes many features that make the database easier to manage. Oracle management fundamentally improved with the introduction of Oracle Database 10*g*, and has continued to evolve toward being more self-tuning and self-managing in subsequent database releases. If you are still managing Oracle Databases using older techniques (e.g., scripts), you might want to reevaluate your thinking on management.

Since Oracle Database 10*g*, statistics are automatically gathered to an Automatic Workload Repository (AWR) within the database. Oracle's Automatic Database Diagnostic Monitor (ADDM) evaluates the statistics on a regular basis and sends alerts of potential problem conditions to Oracle Enterprise Manager, where you can evaluate the condition in more detail and potentially take corrective actions. Some of the newer fully automated features, such as Automatic Memory Management, also leverage data gathered in the AWR.

Oracle has a near real-time view of current database conditions as it makes automated recommendations. Such recommendations will often be more accurate than would be possible with the manual processes you might have used in the past. In the following subsections, we'll introduce the impact this has on Oracle Enterprise Manager and add-on packs, Information Lifecycle Management, backup and recovery, and database availability. We'll also describe the impact of pluggable databases in this section.

Oracle Enterprise Manager 12*c*

Many Oracle Database generations ago, Oracle Enterprise Manager (EM) was introduced as an Oracle Database manager framework. Today, Enterprise Manager 12*c* continues to be a framework used for managing the database, but it is also used for managing Fusion Middleware, Oracle's Applications, Oracle's engineered systems, Cloud-based infrastructure, and more. The framework consists of management services, monitoring, configuration management, task automation, security, and plug-ins for the managed platforms supported. Later in this book, we'll focus our discussion on managing the database deployed to a traditional infrastructure and also to a Cloud-based infrastructure. We'll also discuss the added capabilities provided by Enterprise Manager when managing the Oracle Exadata Database Machine including the monitoring of hardware alerts, configuration management, and proactive support.

Enterprise Manager is accessed via a browser or mobile device (e.g., iOS devices were supported when this edition of the book was published). Enterprise Manager provides a basic interface for monitoring and management of Oracle Database users and user privileges, database schema, and database configuration, status, and backup and recovery. The optional Enterprise Manager Packs that Oracle offers extend the management capabilities:

Oracle Diagnostic Pack for Oracle Database
: The Oracle Diagnostic Pack for the Database provides an automatic and real-time performance interface to the Automatic Database Diagnostic Monitor (ADDM), an automatic workload capture interface to the Database Automatic Workload Repository (AWR), performance comparisons versus ASR baselines, active session history, system monitoring and notification, and Exadata-specific lights-out monitoring and management of nodes, Exadata Storage Server cells, and InfiniBand switches.

Oracle Tuning Pack for Oracle Database
: The Oracle Tuning Pack for the Database provides real-time SQL monitoring used in identifying long-running SQL, a SQL Tuning Advisor that provides recommendations to administrators or that can be run in automatic mode, and a SQL Access Advisor that provides advice on schema design.

Oracle Database Lifecycle Management Pack
: The Oracle Database Lifecycle Management Pack automatically discovers database and application physical servers, enables deployment procedures for provisioning and patching of databases, provides capability to perform patching lifecycle change management, configuration management, and compliance management, and provides Site Guard to manage disaster recovery through integration with Data Guard and filesystem data storage.

Oracle Cloud Management Pack for Oracle Database
: The Cloud Management Pack enables identification of pooled resources, configuration of role-based access, definition of service catalogs and chargeback plans, and supports user requested database resource requests and provisioning where the database is deployed as a service (DBaaS).

Oracle Data Masking Pack
: The Oracle Data Masking Pack enables scanning of an Oracle Database for sensitive data based on patterns, uses referential relationships when determining data elements that should be masked, provides a library of typical masks needed, and supports advanced masking techniques such as condition-based masking, compound masking, deterministic masking, and key-based reversible masking.

Oracle Test Data Management Pack
: The Oracle Test Data Management Pack automatically discovers data relationships and table types, storing them in Application Data Models, and is used to create test databases from production databases.

Real Application Testing Option

Oracle Database 11*g* introduced the capability to rerun production workloads and test system changes through the Real Application Testing Option. This database option includes a Database Replay facility and the SQL Performance Analyzer (SPA). Database Replay captures production workload information, including concurrency, dependencies, and timing. It transforms the workload capture files into replay files, provides a Replay Client for processing the replay files, and provides the means to report on performance statistics and any errors that might occur. The SQL Performance Analyzer captures a SQL workload to be analyzed, measures the performance before database changes and afterward, and identifies performance changes among SQL statements.

Pluggable Databases

As of Oracle Database 12*c*, the database can function as a multitenant container database (CDB) where the Multitenant Option is licensed and hold one or more pluggable databases (PDBs). A PDB is a portable collection of schemas and schema objects that could be unplugged from one CDB and into another. Since PDBs share resources provided by the CDB, this increases the number of databases that can be deployed on a given hardware platform due to more efficient utilization of the resources.

CDBs and PDBs are an important piece of Oracle's DBaaS Cloud strategy as they enable more rapid provisioning of platforms. Key management tasks are simplified since they are managed through the CDBs and related to PDBs. Examples of management features in the CDBs include Active Session History (ASH), alerts, automated database maintenance tasks, ADDM, automatic statistics optimizer collection, Automatic Segment

Advisor, AWR, SQL Management Base (SMB), SPA, SQL Tuning Advisor, and SQL Tuning Sets (STS).

Storage Management

Automatic Storage Management (ASM) is provided as part of the Oracle Database. First introduced with Oracle Database 10*g*, ASM is used to manage pools of storage in designated disk groups that store the database files. The database data is evenly distributed (striped) across disks in a disk group for optimal performance. Data is typically mirrored using ASM for availability. Because an ASM interface is provided through Enterprise Manager, the database administrator can perform this critical management task.

Data in large Oracle Databases is often partitioned to provide a higher degree of manageability and availability. For example, you can take individual partitions offline for maintenance while other partitions remain available for user access. This partitioning capability is provided by the Partitioning Option and was introduced for Oracle8. Since then, the types of partitioning supported have continued to grow in sophistication.

In data warehousing implementations, partitioning is sometimes used to implement rolling windows based on date ranges. Other partitioning types include hash partitioning (used to divide data into partitions using a hashing function and providing an even distribution of data), list partitioning (enables partitioning of data based on discrete values such as geography), interval partitioning (used to automatically create new fixed ranges as needed during data insertions), reference partitioning (where a child table inherits the partitioning strategy of the parent table), and virtual column partitioning (defined by an expression linked to one or more columns). Many of these partitioning types can be used in combination as "composite" partitions. Examples of composite partitions in Oracle Database 12*c* include range-range, range-hash, range-list, list-range, list-hash, and list-list, hash-hash, and interval-reference.

One of the goals of storage management is to minimize the amount of physical disk required. Compression techniques are often combined with partitioning strategies as it often makes sense to compress data on older nonchanging partitions to avoid performance hits during updates. Since Oracle9*i* Release 2, the Oracle Database has included basic compression that typically provides two to four times compression for read-only or inactive tables and partitions. The Advanced Compression Option provides two to four times compression for OLTP databases where updates occur and has been available since Oracle Database 11*g*. Hybrid Columnar Compression is available for Exadata Storage and provides about 10 times compression for data warehouses and 15 times compression for archiving data.

High Availability

Oracle defines a Maximum Availability Architecture (MAA) that addresses recovery time objectives, recovery point objectives, and service level agreement (SLA) objectives

in an organization. Organizations use these guidelines and software capabilities to deliver highly available databases by minimizing planned downtime for maintenance tasks and reducing or eliminating unplanned downtime from outages. Features in the Oracle Database that help manage planned downtime for systems changes, data changes, and applications changes include the ability to do online reconfiguration with rolling upgrades, online redefinition, and edition-based redefinition.

Eliminating unplanned downtime focuses on two areas: data availability and server availability. Solutions for eliminating unplanned data outages include Recovery Manager (RMAN) for backup and recovery, Oracle Secure Backup, the Data Recovery Advisor, Flashback, ASM, Data Guard, and GoldenGate. Real Application Clusters (RAC) is the critical component that helps eliminate unplanned server availability. We've introduced a few of these capabilities previously in this chapter, so we focus on introducing recoverability features and RAC in this section.

Oracle Database 12*c* introduces Application Continuity for masking lost database sessions from users during planned and unplanned downtime. A JDBC replay driver intercepts execution errors when sessions are lost and saves the context of each SQL and PL/SQL call it is instructed to hold. The JDBC replay driver replays the calls when the session is reestablished as directed by a continuity director in the database.

Flashback

The Oracle Database features a variety of Flashback technologies that enable rapid recovery from human errors. A Flashback Query enables a query designating a point in time in the past to be submitted to see how data looked at that time and allows you to identify possible corruption that occurred since. The Flashback Version query enables looking at how a row of data changed over a time interval. A Flashback Transaction Query enables the administrator to see changes made by an individual transaction. Where dependencies exist, the Flashback Transaction capability can be used to back out all changes, relying on undo and archived redo logs.

When individual tables need to be recovered at a previous point in time, Flashback Table is used. Flashback Drop enables easy recovery of dropped tables. Flashback Database enables recovery of an entire Oracle Database to a previous point in time, relying on Flashback logs in the database to restore blocks that have changed.

Recovery Manager

As every database administrator knows, backing up a database is a rather mundane but necessary task. An improper backup makes recovery difficult, if not impossible. Unfortunately, people often realize the extreme importance of this everyday task only after losing business-critical data resulting from a failure of a related system.

Typical kinds of backups include complete database backups, tablespace backups, datafile backups, control file backups, and archivelog backups. Oracle's Recovery Manager (RMAN) enables server-managed backup and recovery of the database and leverages a

Recovery Catalog stored in the database. RMAN can automatically locate, back up, restore, and recover datafiles, control files, and archived redo logs. During backups, RMAN verifies all data blocks to ensure that corrupt blocks are not propagated to backup files. Efficient recovery can occur at the individual block level.

RMAN can restart backups and restore and implement recovery window policies when backups expire. A variety of compression levels are supported to assure reasonable performance where network bottlenecks or CPU limitations exist. A Fast Recovery Area (FRA) can be defined on a file system or ASM disk group enabling better space management. Oracle Enterprise Manager provides a GUI-based interface to RMAN and includes a job scheduler that can be used with RMAN for managing automatic backups to disk.

RMAN can perform incremental backups of Enterprise Edition Databases. Incremental backups will back up only the blocks modified since the last backup of a datafile, tablespace, or database; thus, they're smaller and faster than complete backups. RMAN can also perform point-in-time recovery, which allows the recovery of data until just prior to an undesirable event (such as the mistaken dropping of a table).

Oracle Secure Backup

Various media-management software vendors leverage Oracle's RMAN as part of their backup solutions. Since Oracle Database 10*g*, Oracle has offered a tape backup solution integrated with RMAN for database and filesystem data named Oracle Secure Backup (OSB). OSB features policy-based and fine-grained control over backup media and to the Cloud through support of encryption and key management. OSB also supports tape duplication and managing the rotation of tapes among multiple sites.

Data Guard

Oracle first introduced a standby database feature in Oracle 7.3. A standby database provides a copy of the production database to be used if the primary database is lost—for example, in the event of primary site failure or during routine maintenance. Primary and standby databases may be geographically separated. The standby database is created from a copy of the production database and updated through the application of archived redo logs generated by the production database. Since Oracle9*i*, Data Guard fully automates this process, including the copying and applying of logs. Agents are deployed on both the production and standby database, and a Data Guard Broker coordinates commands. A single Data Guard command invokes the steps required for failover.

Oracle Database 10*g* introduced support for real-time application of redo data, integration with the Flashback Database feature, archivelog, and support of rolling upgrades. The Active Data Guard Option introduced with Oracle Database 11*g* enabled a standby database to be used for queries, sorting, and reporting even as changes from the production system are being applied. Oracle Database 12*c* introduces a lightweight Oracle instance Far Sync standby (with no datafiles) that is used to reliably forward redo

synchronously to one or more remote locations and greatly widen distances where Data Guard might be deployed.

In addition to providing physical standby database support, Data Guard can be used to create a logical standby database where Oracle archive logs are transformed into SQL transactions and applied to the open standby database. Data Guard also supports snapshot standbys where redo is received but not applied when data is simply to be replicated to the standby for testing purposes (such standbys can be converted to physical standbys, and then redo is applied).

If an outage occurs, the Data Recovery Advisor leverages RMAN and Data Guard (including standbys) in determining the best recovery options that minimize any data loss. Administrators can choose among these options or the Advisor can be set to automatically run the best choice.

Fail Safe

The Fail Safe feature provides a higher level of reliability for an Oracle Database on a Windows platform than simple high-availability approaches in leveraging Microsoft Cluster Services. Failover is implemented through a second system or node that provides access to data residing on a shared disk when the first system or node fails. Fail Safe is primarily a disaster recovery tool, so some downtime does occur as part of a failover operation. The recommended solution for high server availability on all platforms, including Windows, is RAC.

Oracle Real Application Clusters

RAC first appeared as an option for Oracle9*i*, replacing the Oracle Parallel Server (OPS) option. RAC can provide failover support as well as increased scalability on Unix operating system variations, Linux, and Windows clusters. Key to RAC's improved scalability was the introduction of Cache Fusion that greatly minimized the amount of writing to disk that was formerly used to control data locks. Oracle Database 10*g* introduced a new level of RAC portability and Oracle support by providing integrated "clusterware" for supported RAC platforms.

With Real Application Clusters, you can deploy multiple Oracle instances on multiple nodes of a clustered solution or in a grid configuration. RAC coordinates traffic among the systems or nodes, allowing the instances to function as a single database. As a result, the database has proven to scale across dozens of nodes. Since the cluster provides a means by which multiple instances can access the same data, the failure of a single instance will not cause extensive delays while the system recovers. You can simply redirect users to another instance that's still operating. Applications can leverage the Oracle Call Interface (OCI) to provide failover to a second instance transparently to the user.

Data Guard can be used to provide automated failover with bounded recovery time in conjunction with Oracle Real Application Clusters. In addition, it provides client

rerouting from the failed instance to the instance that is available with fast reconnect and automatically captures diagnostic data.

Database Security Features

Oracle includes basic security for managing user access through roles and privileges. These can be managed through Enterprise Manager on a local basis or on a global basis by leveraging Oracle's enterprise user security, a feature in the Advanced Security Option.

Database security features allow you to implement a Virtual Private Database (VPD) using Oracle by creating and attaching policies to database tables, views, or synonyms. These policies are then enforced by placing a predicate WHERE clause on SELECT, INSERT, UPDATE, DELETE, and/or INDEX statements.

New in Oracle Database 12*c*, you can redact or mask data queried by users or applications, taking into account assigned privileges. Full data redaction, partial data redaction, or random data redaction of specified columns in tables or views is supported.

Many organizations face the need to meet more stringent compliance requirements for improved data protection, although database usage now can extend beyond organizational boundaries. Oracle has added several options to the database to enable secure deployment in such challenging environments. These options include the Advanced Security Option, Label Security Option, Database Vault Option, and Audit Vault and Database Firewall Option.

Advanced Security Option

The Advanced Security Option (ASO) enables data encryption of tablespaces and columns in the database via Transparent Data Encryption (TDE), which encrypts and decrypts data without requiring any code in the applications that access this data. Data encrypted in TDE remains encrypted when backed up using RMAN. ASO also provides strong authentication services to the database through two-tier key management consisting of a master encryption key and one or more data encryption keys. Oracle Database 12*c* further enhanced the range of TDE key management capabilities available.

Standards-based network encryption is provided with authentication to the database through Kerberos, KPI, or RADIUS. Industry standard network encryption, enabling more secure Oracle Net connections, includes support for the Advanced Encryption Standard (AES) and the U.S. Triple Data Encryption Standard (3DES).

Label Security Option

Oracle Label Security controls access to data by comparing labels assigned to rows of data with label authorizations granted to users through their privileges. Multiple

authorization levels are possible within a single database. Label security is a higher level interface to row-level security supported in Enterprise Edition.

Label security policies, data and user labels, and protected tables can all be managed through Oracle Enterprise Manager and can also be integrated with Oracle Identity Management. Since Policies are enforced in the database instead of through views, which greatly simplifies management of data accessibility and provides a more secure implementation.

Database Vault Option

The Oracle Database Vault Option allows for another dimension of database security. A standard problem with database security stems from the need for database administrators to have full access to the data they manage—a potential security hole. The Database Vault Option allows you to restrict access granted with system-wide privileges, such as preventing administrators read or write access to data, or restricting administrative access to a defined realm of data, allowing for finer grained separation of administrative duties frequently necessary as databases are consolidated. A security administrator can set factors to define access to the database including Oracle commands available to the different classes of users and administrators and audit-specific dimensions of security. At a more granular level, realms can be defined for limiting access to specific database schemas and roles.

Audit Vault and Database Firewall Option

The Audit Vault and Database Firewall Option includes the Audit Vault Server, Audit Vault Collection Agent, Database Firewall, and Database Firewall Management Server. The Oracle Audit Vault Server monitors Oracle Database audit tables and audit information from other database brands, redo logs, and operating system audit files for suspicious activities and is used to manage audit activities. It includes pre-built compliance reports and entitlement reports for Oracle Databases that show users, roles, and privileges and can send alerts showing where unusual or unauthorized activity is occurring.

The Database Firewall is used to monitor SQL statements transmitted to the Oracle Database and determine whether to allow, log, alert, substitute, or block the SQL. SQL statements from specific users or IP addresses of specific types can be blocked. Database Firewall events are logged to the Audit Vault Server.

Oracle Database Development Tools

Many Oracle tools are available to developers to help present data and build more sophisticated Oracle Database applications. As this book focuses on the Oracle Database, this section briefly describes the main Oracle tools used for database development today: Oracle SQL Developer and Oracle Application Express. Other legacy tools, such as

Oracle Forms Developer, Oracle Designer, and Oracle Programmer, are used with much less frequency today.

Oracle SQL Developer

Oracle SQL Developer is an IDE for any currently supported Oracle Database version that you can download from the Oracle Technology Network at no charge and run on your Windows, Linux, or Apple MAC OS X workstation. With SQL Developer, you can create connections to Oracle Databases, browse database objects, create and modify database objects, query and update data, export data and DDL, import data, process commands, and run and create reports. The product's tools support the editing, debugging, and running of PL/SQL scripts and the DBA Console can be used to manage the database. In addition, SQL Developer can be pointed at non-Oracle databases to view their database objects and data, and it provides capabilities to migrate to an Oracle Database.

The SQL Developer Data Modeler provides a graphical user tool for creating, browsing, and editing database models. Data dictionaries can be imported from Oracle Databases, IBM DB2, and Microsoft SQL Server. Data models can be imported from the Computer Associates ERwin product and Oracle's previous generation design tool, Oracle Designer.

Oracle Application Express

Oracle Application Express (APEX) is an in-Oracle Database rapid development tool freely available for all current editions of the Oracle Database. The tool was designed to create HTML-based applications through a wizard-based approach and has proven to be fairly popular in the Oracle community. APEX has come with the Oracle Database since version 10*g* R2 and is available as a download from the Oracle Technology Network forum.

Developers using APEX access a browser-based declarative development framework and are presented with wizards and property sheets for declaratively building and maintaining applications. Key components include an application builder, SQL workshop, team development environment, and administration and analytics interface. Typical use cases include creation of data-driven applications, SQL-based reporting, spreadsheet conversion to web applications, Oracle Forms applications modernization to HTML and Web 2.0, and Microsoft Access replacement.

APEX provides flexibility for development and deployment in a variety of scenarios including local, on multitenant private Clouds (with workspaces for each department and self-service provisioning), or on public Clouds such as the Oracle Database Cloud Service. The framework is flexible in how applications might be accessed and in 2011 added support for mobile devices (leveraging jQuery Mobile) and HTML5.

Other Oracle Databases

Today, Oracle offers other databases including Oracle MySQL, Berkeley DB, Oracle NoSQL Database, and TimesTen. We'll also briefly touch upon the Cloudera Hadoop distribution included with Oracle's Big Data Appliance. These database engines have unique code lines with different intended roles. There are entire books written about these alternative databases. For this reason, we will describe these briefly in the following subsections but will not explore their capabilities in detail elsewhere in this book.

Oracle MySQL

The MySQL database is an open source relational database, with source code made available under the GNU General Public License (GPL). Development of MySQL began in 1994 by Michael Widenius and David Axmark and it was first released in 1995. Over time, the MySQL database grew into the most popular open source database. MySQL was acquired by Sun in 2008, and Sun was then acquired by Oracle in 2010. In addition to the MySQL Database, there is a free graphical data modeling, SQL development, and administration tool called the MySQL Workbench that is often deployed as part of the infrastructure. Oracle bundles the Workbench with its editions of MySQL.

A MySQL Community Edition remains freely downloadable today and is supported by open source developers. Oracle offers support for several editions, including a Classic Edition, Standard Edition, Enterprise Edition, and Cluster Carrier Grade Edition. The editions share in common a reputation for rapid installation, low total cost of ownership, and exceptional scalability and performance for many applications.

The Classic Edition is intended to be used as an embedded database by OEMs, ISVs, and VARs developing read-intensive applications using MyISAM. For more demanding OLTP applications, the Standard Edition provides a basic database engine that also includes the InnoDB engine.

As scalability and performance requirements grow, the Enterprise Edition is designed to provide additional functionality. Key components include:

- MySQL Enterprise Backup for hot compressed backups; full, incremental, and partial backups; full and partial restores; and point-in-time recovery
- MySQL Enterprise High Availability, which includes replication, Oracle VM template support, and Solaris and Windows failover clustering
- MySQL Enterprise Scalability for high-demand query and workload
- MySQL Enterprise Security for supporting pluggable authentication, including Windows Active Directory support
- MySQL Enterprise Audit for supporting policy-based auditing in applications

- MySQL Enterprise Monitor for proactive best practices tips and security alerting through the Enterprise Dashboard, Advisors, Replication Monitor, and Query Analyzer (that identifies SQL code slowing query performance)

The MySQL Cluster Carrier Grade Edition automatically partitions database tables across nodes of a multinode commodity hardware platform providing horizontal scalability using the NDB engine. This Edition also enables highly available configurations and supports active-active replication for clustering across distributed geographies, including for purposes of disaster recovery. Nodes can be added and database schema can be updated while the database is online. The MySQL Cluster Manager automates common cluster management tasks and provides extended cluster monitoring support.

Berkeley DB & Oracle NoSQL Database

Oracle Berkeley DB is an extremely small-footprint embedded database engine. The engine supports transactional workloads and features multiversion concurrency control, indexing, encryption, and replication. It comes in variations that Oracle labels as Berkeley DB, Berkeley DB Java Edition, and Berkeley DB XML. Data interfaces supported include the SQLite API, Java Objects, key value, and XQuery/XPath for XML. For example, the Java Edition provides a direct persistence layer (DPL) API for EJB-style persistence and a Java collection API and the database is a single JAR file.

Berkeley DB is designed to be deployed with and run in the same process as your applications. Footprints for the database have static library sizes of less than 1 MB and runtime dynamic memory requirements of a few kilobytes.

When mobile applications deployed using Berkeley DB are to be attached to an Oracle Database, the Oracle Database Mobile Server provides a sync engine and a mobile manager. This server can be deployed on WebLogic or GlassFish. Mobile clients can include Java, Android, Blackberry, Windows, and Linux.

In 2011, Oracle introduced the Oracle NoSQL Database that leverages the Berkeley DB Java Edition engine. It is designed to support large volume and low latency applications and is implemented on a distributed key value engine with transparent load balancing across the nodes it is deployed to. The applications are written specifying data consistency policies depending on the amount of overhead that is acceptable. These policies can range from absolute consistency to time-based consistency to weak consistency (and lowest latency). The NoSQL Database can be deployed as highly available through configurable multiple replicas and for disaster recovery by locating the replicas in secondary locations.

Two editions are available. The Oracle NoSQL Database Community Edition is Oracle's open source offering (AGPL version 3 license). It is included with the Oracle Big Data Appliance. Oracle also offers the Oracle NoSQL Database Enterprise Edition, which is Oracle-supported.

Oracle TimesTen

Oracle TimesTen is a relational database that is stored in physical memory and is typically used where very high-performance transaction-processing workloads are present. Access to the TimesTen database is available through SQL, JDBC, JMS, and ODBC. TimesTen databases can be deployed as exclusive or shared and can be created as permanent or temporary.

The database is refreshed by gathering data using TimesTen libraries deployed to applications or by using a Cache Connect Option to an Oracle Database. Because data is read and updated in memory, average update or read response times are typically measured in the millionths of seconds. The Cache Connect Option supports both read and write caching of Oracle Database data. Updates between TimesTen and Oracle can be bidirectional. When paired with the Oracle Database, this is referenced as the Oracle In-Memory Database Cache (IMDB Cache).

As is typical for embedded databases, TimesTen requires almost no ongoing administration. Replication is possible from one TimesTen database to another through an option and is, by default, asynchronous.

Oracle introduced a variation named TimesTen for Exalytics with the Oracle Exalytics In-Memory Machine in 2011. As might be expected for optimizing performance of the BI Foundation Suite, key analytics and query functionality was added, including OLAP grouping operators, analytic functions, time functions, and columnar compression.

Cloudera Distribution of Hadoop

Apache Hadoop is an open source framework for data-intensive applications where the data is considered to be semi-structured or unstructured. Such data typically comes from sensors, social media feeds, text, and web log data and contains descriptors, data of value tied to those descriptors, and other miscellaneous data. Thus, the data of value is relatively sparse. It was recognized by developers of search engines such as Google and Yahoo! in the early 2000s that there was a need to map such data and reduce it down to data of value to make sense of it. Hence, MapReduce was developed as a programming paradigm and is embedded in Java, Python, and other programming languages. The data itself is stored and analyzed in the Hadoop Distributed File System (HDFS) that is deployed across a cluster of a multinode hardware configuration.

The most popular distribution of Hadoop, at the time this book was published, is the Cloudera Distribution of Hadoop (CDH). CDH is included with Oracle's Big Data Appliance (BDA) and supported by Oracle. In addition to HDFS and MapReduce, CDH includes other Hadoop components including Flume, Fuse-DFS, HBase, Hive, Mahout, Oozie, Pig, Sqoop, and Zookeeper. CDH also provides the Cloudera Manager for managing the Hadoop cluster, and Oracle provides further platform management integration via Enterprise Manager.

As some organizations prefer to make the Oracle Database the source for all data, Oracle Database 12*c* introduced pattern matching within the Oracle Database. This enables organizations with Oracle Database skills to begin MapReduce-like development in a more familiar environment. More commonly, organizations will perform MapReduce on their Hadoop cluster and then will load the data of value into a data warehouse. Oracle offers a Loader for Hadoop (OLH) that optimizes performance when loading this data into an Oracle Database.

CHAPTER 2
Oracle Architecture

This chapter focuses on the concepts and structures at the core of the Oracle Database. When you understand the architecture of the Oracle Database server, you'll have a context for understanding the rest of the features of Oracle described in this book.

An Oracle Database consists of both physical and logical components. The first section of this chapter covers the difference between an Oracle Database and an instance, and subsequent sections describe physical components, the instance, and the data dictionary.

Databases and Instances

Many Oracle practitioners use the terms *instance* and *database* interchangeably. In fact, an instance and a database are different (but related) entities. This distinction is important because it provides insight into Oracle's architecture.

In Oracle, the term *database* refers to the physical storage of information, and the term *instance* refers to the software executing on the server that provides access to the information in the database and the resources that software uses. The instance runs on the computer or server; the database is stored on the disks attached to the server. Figure 2-1 illustrates this relationship.

The database is *physical*: it consists of files stored on disks. The instance is *logical*: it consists of in-memory structures and processes on the server. For example, Oracle uses an area of shared memory called the System Global Area (SGA) and a private memory area for each process called the Program Global Area (PGA). An instance can access one and only one database, although multiple instances can be part of the same database. Instances are temporal, but databases, with proper maintenance, last forever.

Users do not directly access the information in an Oracle Database. Instead, they pass requests for information to an Oracle instance.

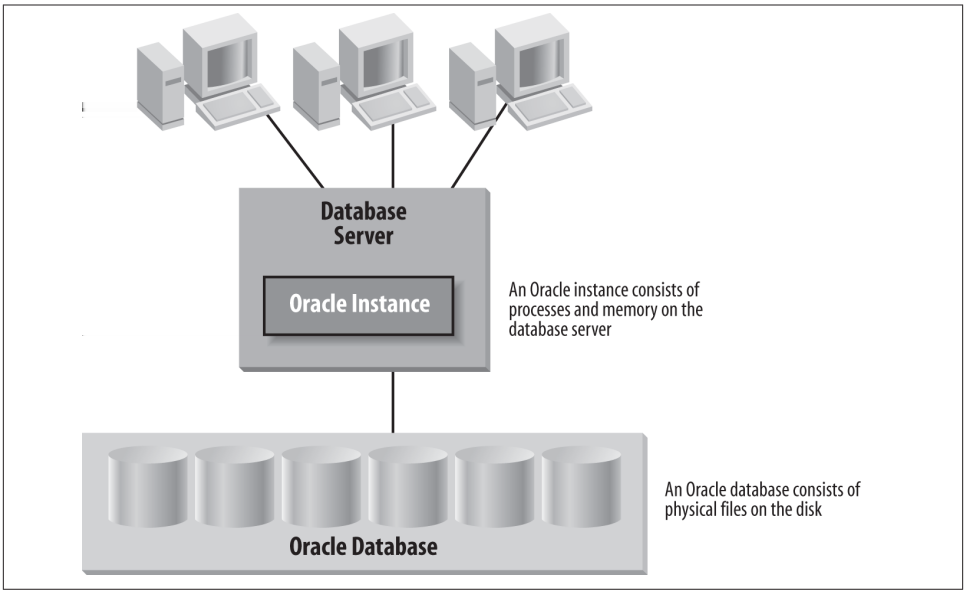


Figure 2-1. An instance and a database

The real world provides a useful analogy for instances and databases. An instance can be thought of as a bridge to the database, which can be thought of as an island. Traffic flows on and off the island via the bridge. If the bridge is closed, the island exists but no traffic flow is possible. In Oracle terms, if the instance is up, data can flow in and out of the database. The physical state of the database can change. If the instance is down, users cannot access the database even though it still exists physically. The database is static: no changes can occur to it. When the instance comes back into service, the data will be there waiting for it.

Pluggable databases, part of a new feature in Oracle Database 12*c* called Oracle Multitenant and described below, give you the ability to isolate some of the logical capabilities of an Oracle instance within the instance itself, but the basic distinction between the database and the instance remains the core concept of Oracle Database architecture.

Oracle Database Structures

Oracle's database structures include schemas, pluggable databases (new in Oracle Database 12*c*), tablespaces, control files, redo logfiles, archived logs, block change tracking files, Flashback logs, and recovery backup (RMAN) files. This section introduces many of the structures and other components that make up a complete database.

Schemas

Schemas are a core part of the logical organization of an Oracle Database. Schemas are matched to an Oracle Database user. When you create a schema, you specify a password for the user, the tablespace(s) for the schema, and the amount of each tablespace available to that schema/user.

The schema is the basic logical isolation unit of the Oracle Database. Table names must be unique only within the context of their schema. The owner of a schema gets access to all data structures within the schema, and access to those objects must be GRANTed specifically to other users. All accessible data structures within a schema can normally only be accessed by other users by adding the name of the schema before the data structure name, although you can create synonyms, discussed in Chapter 4, to provide a common name to other users.

Schemas are also used as the foundation for multitenancy for both Oracle Application Express and the Oracle Database Cloud, both discussed in Chapter 15.

Tablespaces

All of the data stored within an Oracle Database must reside in a tablespace. A *tablespace* is a logical structure; you can't look at the operating system and see a tablespace. Each tablespace is composed of physical structures called *datafiles*; each tablespace must consist of one or more datafiles, and each datafile can belong to only one tablespace. When creating a table, you can specify the tablespace in which to create it. Oracle will then find space for it in one of the datafiles that make up the tablespace.

Figure 2-2 shows the relationship of tablespaces to datafiles for a database.

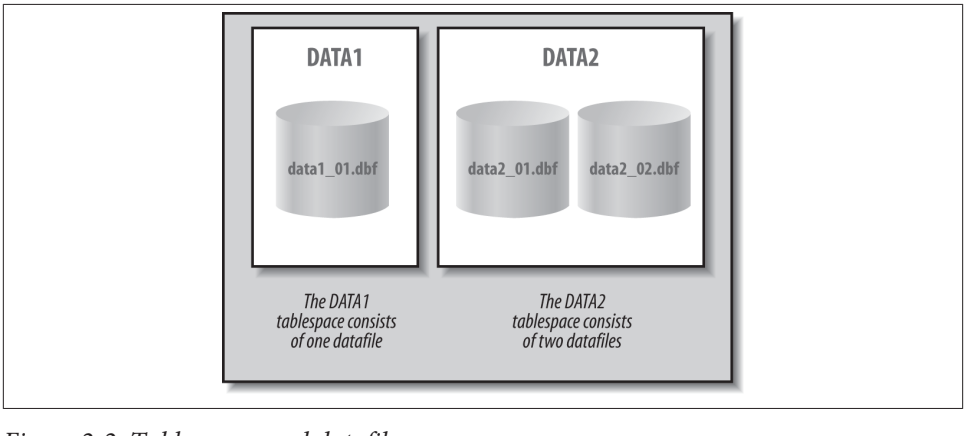


Figure 2-2. Tablespaces and datafiles

This figure shows two tablespaces within an Oracle Database. When you create a new table in this Oracle Database, you may place it in the DATA1 tablespace or the DATA2 tablespace. It will physically reside in one of the datafiles that make up the specified tablespace.

Oracle's default tablespaces for all types of tables are *locally managed tablespaces* as of Oracle Database 10*g* Release 2. As the name implies, locally managed tablespaces are typically more efficient, since the tracking of space in the tablespace is done locally, rather than contending for space management information in the shared data dictionary. Locally managed tablespaces enable creation of *bigfiletablespaces* that can leverage 64-bit systems and their ability to manage ultra-large files.

Oracle9*i* introduced the concept of Oracle Managed Files (OMFs), which enable your database to automatically create, name, and delete, where appropriate, all the files that make up your database. OMFs reduce the maintenance overhead of naming and tracking the filenames for your database, as well as avoiding the problems that can result from human errors in performing these tasks.

Oracle Databases can be deployed on up to 64,000 datafiles. Because a bigfiletablespace can contain a file that is 1,024 times larger than a smallfiletablespace, and bigfiletablespaces have 32 KB block sizes on 64-bit operating systems, the Oracle Database can grow to up to 8 exabytes in size (an exabyte is equivalent to a million terabytes).[1] The bigfiletablespace is designed for use with Oracle's Automatic Storage Management (ASM), other logical volume managers that support striping, and RAID.[2]

Files of a database

There are three fundamental types of physical files that make up an Oracle Database:

- Control files
- Datafiles
- Redo logfiles

These three fundamental types represent the physical database itself. Figure 2-3 illustrates the three types of files and their interrelationships.

1. The ultimate size of a bigfile depends on the limitations of the underlying operating system.
2. RAID stands for "redundant array of inexpensive disks" and is described in Chapter 7.

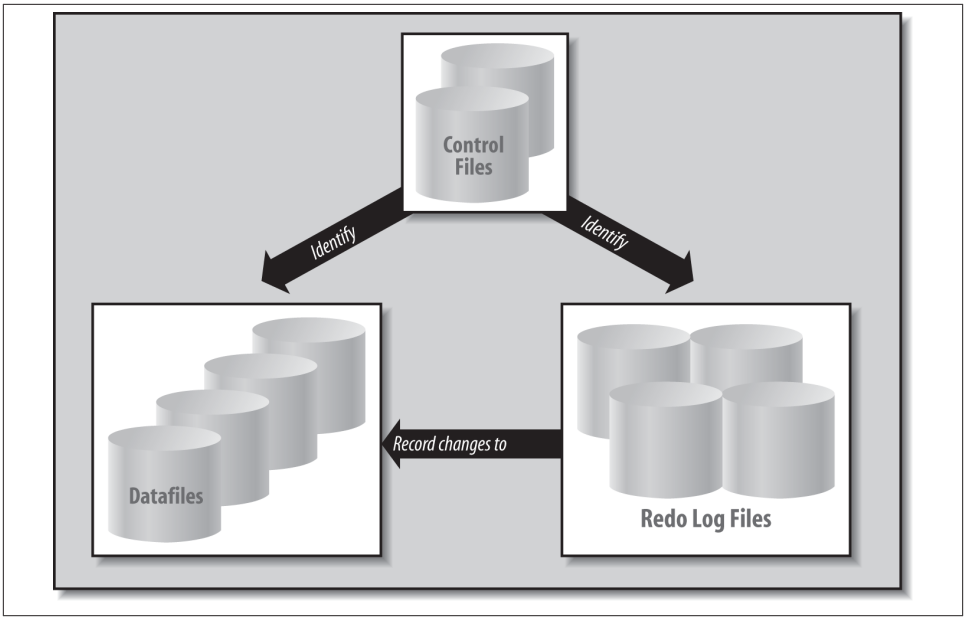


Figure 2-3. The files that make up a database

The control file contains locations for other physical files that form the database: the datafiles and redo logfiles. It also contains key information about the contents and state of the database, including:

- The name of the database
- When the database was created
- Names and locations of datafiles and redo logfiles
- Tablespace information
- Datafile offline ranges
- The log history and current log sequence information
- Archived log information
- Backup set, pieces, datafile, and redo log information
- Datafile copy information
- Checkpoint information

In addition to providing this information at startup, control files are also useful when removing a database. Since Oracle Database 10*g*, the DROP DATABASE command can be used to delete the database files listed in the database control file as well as the control file itself. With Oracle Database 12*c*, the control file indicates whether the database is a

multitenant container database or pluggable database, as well as additional information related to pluggables, which are described in the next section.

Pluggable Databases

All of the structures listed above have been a part of the Oracle Database since the first publication of this book, more than 10 years ago. Oracle Database 12*c* introduces a new structural concept, the *pluggable database*, included as part of a feature called the Multitenant database, which provides another layer of separation within a single database instance.

The concept behind pluggable databases is straightforward—a pluggable database is a layer of isolation between a database instance and a schema. Those operations and entities that apply to an instance are implemented at the level of the multitenant container database (CDB) and operate across multiple separate pluggable databases (PDBs). This separation increases the flexibility of privilege and responsibility assigned to a pluggable database, while not having to also increase the operational overhead associated with an instance. The separation also makes it quicker and easier to create a new pluggable database and to upgrade pluggable databases by simply plugging them into an upgraded CDB.

Schemas live within a PDB. Since a PDB can support multiple schemas, pluggable databases are a good tool to use for database consolidation, since they can support the complete structures of applications with multiple schemas without concern for conflicts within a single database instance. In effect, a PDB is a logically distinct database within an Oracle instance. The CDB can contain users and roles that are common across PDBs. Even better, in terms of consolidation, multiple PDBs share common database background processes, which means that multiple PDBs in a single instance require fewer resources than the multiple isolated instances.

All data and metadata for a PDB are stored within the PDB. This organization means that you will be able to unplug a PDB from one Oracle Database instance and plug it into another instance transparently, simplifying support of multiple environments such as for develop and test purposes. You will even be able to upgrade a PDB by simply moving it from one CDB to another CDB that has been upgraded.

Implementation

Oracle Database 12*c* will allow for a single-tenant version, which includes a single PDB in a CDB, or multiple PDBs with the Oracle Multitenant Option.

All CDBs come with a seed PDB, which is used as the template for other PDBs that are created by an instance administrator. You can move PDBs from one CDB to another, or plug in a non-PDB database to an Oracle Database 12*c* instance that supports PDBs, which will transform it into a PDB. You can also clone an existing PDB to create a copy.

If you define users for the container database, these entities are visible to all PDBs for that container and must be unique for the container and all PDBs. You can define a common user in a container database, who will be visible in all PDBs within that container.

Access to PDBs is defined through the use of services, which are defined at the CDB level and used in SQL*Net connect strings.

Finally, a PDB is completely backwards-compatible with a standard, non-PDB (pre 12*c*) Oracle Database.

PDBs and Oracle features

Oracle Database 12*c* instances are essentially seen as the complete collection of PDBs and the multitenant container database. System-wide processes, like backup, instance recovery, and the use of Data Guard apply to the entire instance. When using a PDB-enabled instance as a node in RAC, the node is seen as the instance, but individual sessions connect to a single PDB.

Since a single Oracle Database instance can support multiple PDBs, you can achieve separation between PDBs without multiplying the amount of overhead required for managing system processes. You can designate administrators for individual PDBs. Some administrative functions, such as point-in-time recovery or Flashback Query, can be applied to individual PDBs, and you can do ad hoc backups for PDBs separately from the CDB. Many system parameters are also settable for each PDB.

Database Resource Manager has been extended to allow for allocation of resources between PDBs, and Enterprise Manager and other tools have been integrated to allow for use with these new structures in Oracle Database 12*c*.

There are a number of implications for Oracle security. A *common user* can be defined for a CDB and must begin with c## or C##. Common users have schemas in one or more PDBs, and those schemas can contain different objects.

You can define *common roles* at the container level in Oracle Database 12*c*. Privileges and roles can be granted locally, for a PDB, to either a local or common user, or commonly, across all PDBs to a common user. You can grant access to users across PDBs through the use of database links, similar to how you would allow access to objects in separate databases. The common role of PUBLIC is defined at the CDB level.

Database Initialization

At Oracle Database instance startup, initialization parameters are read to determine how the database will leverage physical infrastructure and for other key instance configuration information. Initialization parameters are stored in an instance initialization parameter file, often referred to as *INIT.ORA*, or, since Oracle9*i*, in a repository called the server parameter file (or *SPFILE*). The number of initialization parameters that must

be specified has been greatly reduced with each Oracle Database release. Oracle provides a sample initialization file that can be used at database startup, and the Database Configuration Assistant (DCA) prompts you for values that must be provided on a custom basis (such as database name).

The recommended minimum set of initialization parameters that should be specified as of Oracle Database 12*c* include:

CONTROL_FILES
: The control file locations

DB_NAME
: The local database name

MEMORY_TARGET
: The target memory size that is automatically allocated to SGA and instance PGA components

All other initialization parameters are pre-set to default values. As an example in the shift toward automation, since Oracle Database 11*g*, the UNDO_MANAGEMENT parameter default is set to automatic undo management. Undo is used in the rollback of transactions, and for database recovery, read consistency, and Flashback features. (Redo records, though, reside in the physical redo logs; they store changes to data segments and undo segment data blocks, and they hold a transaction table of the undo segments. The different purposes of UNDO and redo are explored over the next few chapters.) The undo retention period, which controls how far back features like Flashback can work, is now self-tuned by Oracle based on how the undo tablespace is configured.

For your database release, check the documentation regarding optional initialization parameters, as these change from release to release. Some of them are described in the following sections.

Deploying Physical Components

This section is not a substitute for Oracle's installation procedures, but it should provide you with some practical guidance as you plan deployment of an Oracle Database.

Note that for Oracle Database 12*c*, there are only one set of control files, redo logfiles, undo files, and temp files for each instance, while datafiles are associated with individual pluggable databases.

Control Files

A database should have at least two copies of the control file on different physical disks. Without a current copy of the control file, you run the risk of losing track of portions of your database. Losing control files is not necessarily fatal—there are ways to rebuild

them. However, rebuilding control files can be difficult, introduces risk, and can be easily avoided.

The location of the control files is defined, as previously mentioned, by the CONTROL_FILES initialization parameter. You can specify multiple copies of control files by indicating multiple locations in the CONTROL_FILES parameter for the instance, as illustrated here:

```
control_files = (/u00/oradata/control.001.dbf,
     /u01/oradata/control.002.dbf,
     /u02/oradata/control.003.dbf)
```

This parameter tells the instance where to find the control files. Oracle will ensure that all copies of the control file are kept in sync so all updates to the control files will occur at the same time. If you do not specify this parameter, Oracle will create a control file using a default filename or by leveraging Oracle Managed Files (if enabled).

Many Oracle Databases are deployed on some type of redundant disk solution such as RAID-1 or RAID-5 to avoid data loss when a disk fails. (RAID is covered in more detail in Chapter 7.) You might conclude that storing the control file on protected disk storage eliminates the need for maintaining multiple copies of control files and that losing a disk won't mean loss of the control file. But there are two reasons why this is not an appropriate conclusion:

1. If you lose more than one disk in a *striped array* or *mirror-pair*, you will lose the data on those disks. This type of event is statistically rare, but if it does occur, you could be faced with a damaged or lost control file. As you would have your hands full recovering from the multiple disk failures, you would likely prefer to avoid rebuilding control files during the recovery process. Multiplexing your control files, even when each copy is on redundant disk storage, provides an additional level of physical security.
2. Redundant disk storage does nothing to protect you from the perpetual threat of human error. Someone could inadvertently delete or rename a control file, copy another file over it, or move it. A mirrored disk will faithfully mirror these actions, and multiplexed control files will leave you with one or more surviving copies of the control file when one of the copies is damaged or lost.

You do not need to be concerned with additional performance impact when writing to multiple control files. Updates to the control files are insignificant compared to other disk I/O that occurs in an Oracle environment.

Datafiles

Datafiles contain the actual data stored in the database, the tables and indexes that store data, the data dictionary that maintains information about these data structures, and the rollback segments used to implement multiuser concurrency.

A datafile is composed of Oracle Database blocks that, in turn, are composed of operating system blocks on a disk. Oracle block sizes range from 2 KB to 32 KB. Prior to Oracle9*i*, only a single block size could be present in the entire database. In versions of the database since the introduction of Oracle9*i*, you still set a default block size for the database, but you can also have up to five other block sizes in a database (though only a single block size for each tablespace). Figure 2-4 illustrates the relationship of Oracle blocks to operating system blocks.

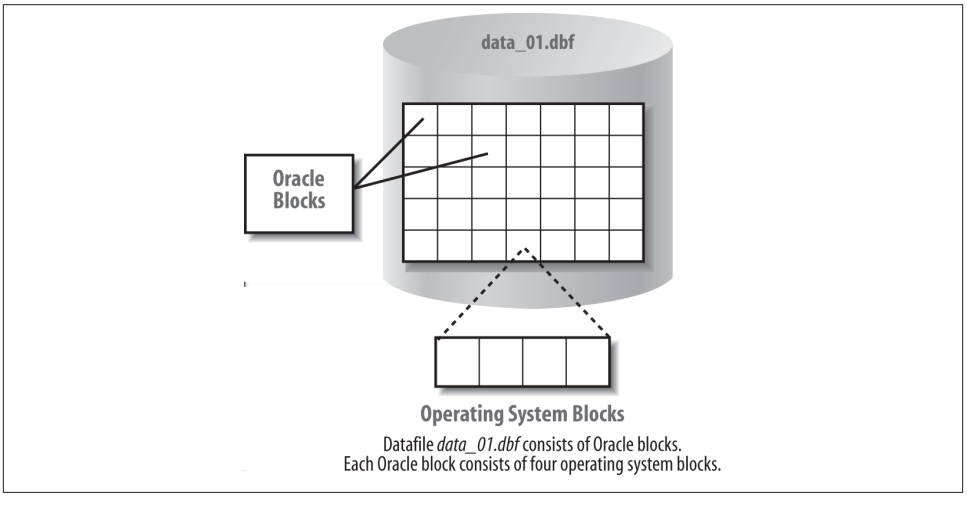


Figure 2-4. Oracle blocks and operating system blocks

Datafiles belong to only one database and to only one tablespace within that database. Data is read in units of Oracle blocks from the datafiles into memory as needed, based on the work users are doing. Blocks of data are written from memory to the datafiles stored on disk as needed to ensure that the database reliably records changes made by users.

Datafiles are the lowest level of granularity between an Oracle Database and the operating system. When you lay out a database on the I/O subsystem, the smallest piece you place in any location is a datafile. Tuning the I/O subsystem to improve Oracle performance typically involves moving datafiles from one set of disks to another. Automatic Storage Management, included in Oracle Databases since Oracle Database 10*g*, provides automatic striping and eliminates manual effort for this tuning task.

Setting the Database Block Size

Prior to Oracle9*i*, you set the database block size for an Oracle Database at the time you created the database, and you couldn't change it without re-creating the database. Since Oracle9*i*, you have more flexibility, because you can have multiple block sizes in the same database. In all versions, the default block size for the database is set using the DB_BLOCK_SIZE instance initialization parameter.

How do you choose an appropriate block size for an Oracle Database? Oracle defaults to a block size based on the operating system used, but understanding the implications of the block size can help you determine a more appropriate setting for your workload.

The block size is the minimum amount of data that can be read or written at one time. In online transaction processing (OLTP) systems, a transaction typically involves a relatively small, well-defined set of rows, such as the rows used for placing an order for a set of products for a specific customer. The access to rows in these operations tends to be through indexes, as opposed to through a scan of the entire table. Because of this, having smaller blocks (4 KB) might be appropriate. Oracle won't waste system resources by accessing larger blocks that contain additional data not required by the transaction.

Data warehouses workloads can include reading millions of rows and scans of all the data in a table. For this type of activity, using bigger database blocks enables each block read to deliver more data to the requesting user. To support these operations best, data warehouses usually have larger blocks, such as 8 KB or 16 KB. Each I/O operation might take a little longer due to the larger block size, but the reduced number of operations will end up improving overall performance.

Datafile structure

The first block of each datafile is called the *datafile header*. It contains critical information used to maintain the overall integrity of the database. One of the most critical pieces of information in this header is the *checkpoint structure*. This is a logical timestamp that indicates the last point at which changes were written to the datafile. This timestamp is critical during an Oracle recovery process as the timestamp in the header determines which redo logs to apply in bringing the datafile to the current point in time.

Extents and segments

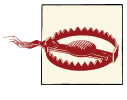
From a physical point of view, a datafile is stored as operating system blocks. From a logical point of view, datafiles have three intermediate organizational levels: data blocks, extents, and segments. An *extent* is a set of data blocks that are contiguous within an Oracle datafile. A *segment* is an object that takes up space in an Oracle Database, such as a table or an index that is composed of one or more extents.

When Oracle updates data, it first attempts to update the data in the same data block. If there is not enough room in the data block for the new information, Oracle will write the data to a new data block that could be in a different extent.

For more information on segments and extents and how they affect performance, refer to the section on "What About Database Fragmentation?" on page 145 in Chapter 5. This discussion is especially important if you are running an older release of Oracle. Oracle Database 10*g* added a Segment Advisor that greatly simplifies reclaiming unused space in current database versions.

Redo Logfiles

Redo logfiles contain a "recording" of the changes made to the database as a result of transactions and internal Oracle activities. Since Oracle usually caches changed blocks in memory, when instance failure occurs, some changed blocks might not have been written out to the datafiles. The recording of the changes in the redo logs can be used to play back the changes lost when the failure occurred, thus protecting transactional consistency.



These files are sometimes confused with rollback buffers supporting concurrency, described in Chapter 8. They are not the same!

In addition, redo logfiles are used for "undo" operations when a ROLLBACK statement is issued. Uncommitted changes to the database are rolled back to the database image at the last commit.

Suppressing Redo Logging

By default, Oracle logs all changes made to the database. The generation of redo logs adds a certain amount of overhead. You can suppress redo log generation to speed up specific operations, but doing so means the operation in question won't be logged in the redo logs and you will not be able to recover that operation in the event of a failure.

If you do decide to suppress redo logging for certain operations, you would include the NOLOGGING keyword in the SQL statement for the operation. (Note that prior to Oracle8, the keyword was UNRECOVERABLE.) If a failure occurred during the operation, you would need to repeat the operation. For example, you might build an index on a table without generating redo information. In the event that a database failure occurs and the database is recovered, the index will not be re-created because it wasn't logged. You'd simply execute the script originally intended to create the index again.

To simplify operations in the event of a failure, we recommend that you always take a backup after an unlogged operation if you cannot afford to lose the object created by the operation or you cannot repeat the operation for some reason. In addition to using the NOLOGGING keyword in certain commands, you can also mark a table or an entire tablespace with the NOLOGGING attribute. This will suppress redo information for all applicable operations on the table or for all tables in the tablespace.

Multiplexing redo logfiles

Oracle defines specific terminology to describe how it manages redo logs. Each Oracle instance uses a *thread* of redo to record the changes it makes to the database. A thread of redo is composed of redo log groups, which are composed of one or more redo log members.

Logically, you can think of a redo log group as a single redo logfile. However, Oracle allows you to specify multiple copies of a redo log to protect the all-important integrity of the redo log. By creating multiple copies of each redo logfile, you protect the redo logfile from disk failure and other types of disasters.

Figure 2-5 illustrates a thread of redo with groups and members. The figure shows two members per group, with each redo log mirrored.

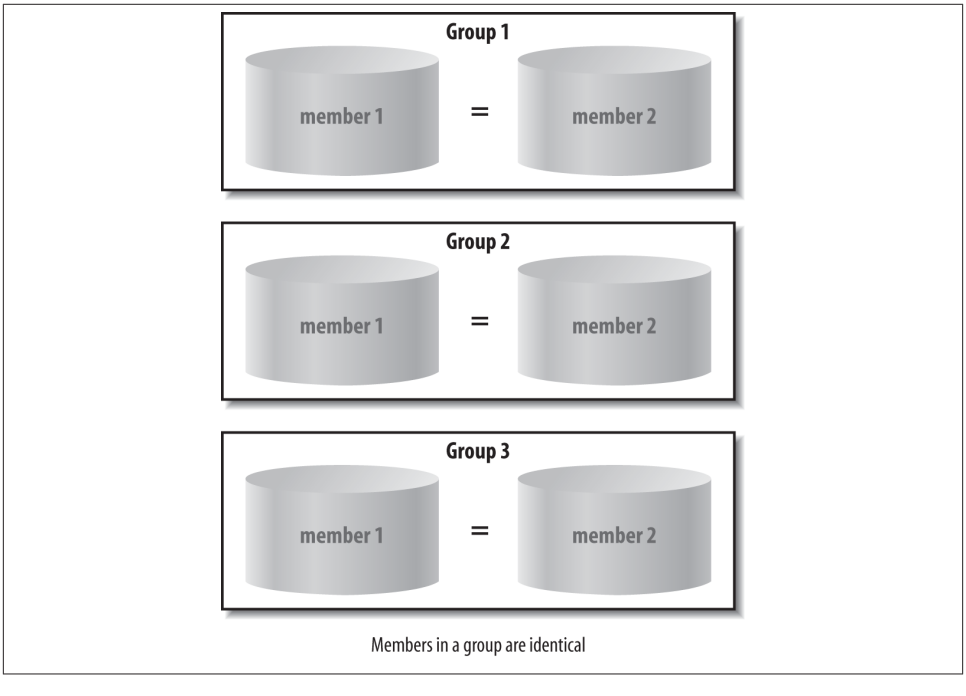


Figure 2-5. A thread of redo

When multiple members are in a redo log group, Oracle maintains multiple copies of the redo logfiles. The same arguments used for multiplexing of control files apply here. However, though you can rebuild the static part of a control file if you lose it, there is no way to reproduce a lost redo logfile. So, be sure to have multiple copies of the redo file. Simple redundant disk protection is not sufficient for cases in which human error results in the corruption or deletion of a redo logfile.

Oracle writes *synchronously* to all redo log members. Oracle will wait for confirmation that all copies of the redo log have been successfully updated on disk before the redo write is considered done. If you put one copy on a fast or lightly loaded disk, and one copy on a slower or busier disk, your performance will be constrained by the slower disk. Oracle has to guarantee that all copies of the redo logfile have been successfully updated to avoid losing data.

Consider what could happen if Oracle were to write multiple redo logs asynchronously, writing to a primary log and then updating the copies later in the background. If a failure occurs that brings the system down and damages the primary log, Oracle might not have completed updating all the logs. At this point you have committed transactions that are lost—the primary log that recorded the changes made by the transactions is gone, and the copies of the log are not yet up to date with those changes. To prevent this from occurring, Oracle always waits until all copies of the redo log have been updated.

How Oracle uses the redo logs

Once Oracle fills one redo logfile, it automatically begins to use the next logfile. When the server cycles through all the available redo logfiles, it returns to the first one and reuses it. Oracle keeps track of the different redo logs by using a redo log sequence number. This sequence number is recorded inside the redo logfiles as they are used.

To understand the concepts of redo log filenames and redo log sequence numbers, consider three redo logfiles called *redolog1.log*, *redolog2.log*, and *redolog3.log*. The first time Oracle uses them, the redo log sequence numbers for each will be 1, 2, and 3, respectively. When Oracle returns to the first redo log—*redolog1.log*—it will reuse it and assign it a sequence number of 4. When it moves to *redolog2.log*, it will initialize that file with a sequence number of 5.

Remember that the operating system uses the redo logfile to identify the physical file, while Oracle uses the redo logfile sequence number to determine the order in which the logs were filled and cycled. Because Oracle automatically reuses redo logfiles, the name of the redo logfile is not necessarily indicative of its place in the redo logfile sequence.

Figure 2-6 illustrates the filling and cycling of redo logs.

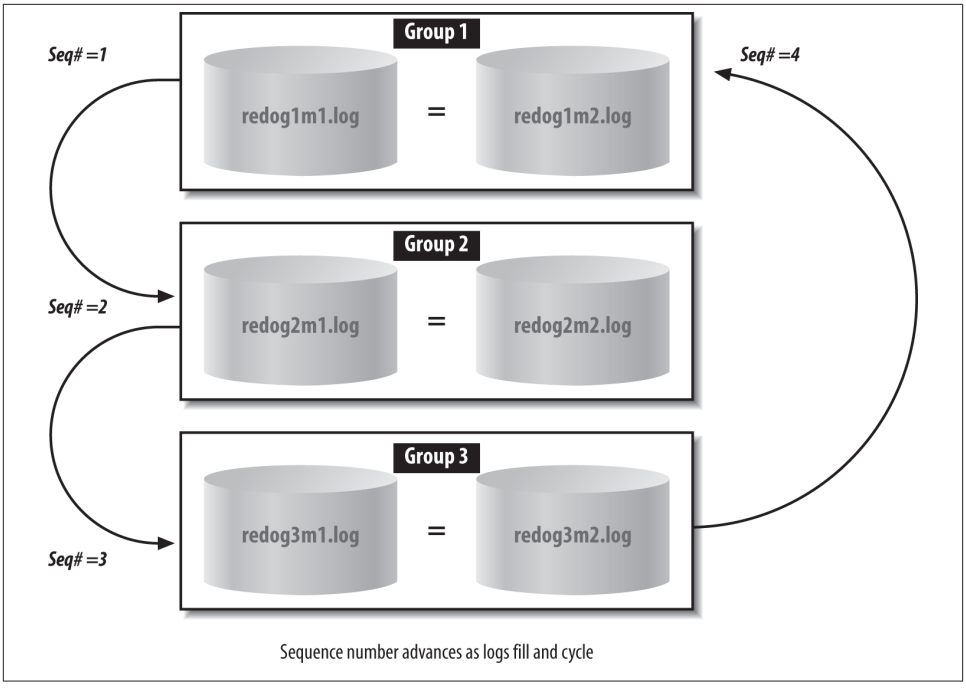


Figure 2-6. Cycling redo logs

Naming conventions for redo logs

The operating system names for the various files that make up a database are very important—at least to humans, who sometimes have to identify these files by their names. If you are not using Oracle Managed Files, you should use naming conventions that capture the purpose and some critical details about the nature of the file. Here's one possible convention for the names of the actual redo logfiles shown in Figure 2-6:

```
redog1m1.log, redog1m2.log, ...
```

The `redo` prefix and `.log` suffixes indicate that this is redo log information. The `g1m1` and `g1m2` character strings capture the group and member numbers. This convention is only an example; it's best to set conventions that you find meaningful and stick to them.

Archived redo logs

You may be wondering how to avoid losing the critical information in the redo log when Oracle cycles over a previously used redo log.

There are actually two ways to address this. The first is quite simple: you don't avoid losing the information and you suffer the consequences in the event of a failure. You will lose the history stored in the redo file when it is overwritten. If a failure occurs that

damages the datafiles, you must restore the entire database to the point in time when the last backup occurred. Since no redo log history exists to reproduce the changes made since the last backup occurred, you will lose the effects of those changes. Very few Oracle shops make this choice, because the inability to recover to the point of failure is unacceptable—it results in lost data.

The second and more practical way to address the issue is to archive the redo logs as they fill. To understand archiving redo logs, you must first understand that there are actually two types of redo logs for Oracle:

Online redo logs
: The operating system files that Oracle cycles through to log the changes made to the database

Archived redo logs
: Copies of the filled online redo logs made to avoid losing redo data as the online redo logs are overwritten

An Oracle Database can run in one of two modes with respect to archiving redo logs:

NOARCHIVELOG
: As the name implies, no redo logs are archived. As Oracle cycles through the logs, the filled logs are reinitialized and overwritten, which erases the history of the changes made to the database. This mode essentially has the disadvantage mentioned above, where a failure could lead to unrecoverable data.

  Choosing not to archive redo logs significantly reduces your options for database backups, as we'll discuss in Chapter 11, and is not advised by Oracle.

ARCHIVELOG
: When Oracle rolls over to a new redo log, it archives the previous redo log. To prevent gaps in the history, a given redo log cannot be reused until it is successfully archived. The archived redo logs, plus the online redo logs, provide a complete history of all changes made to the database. Together, they allow Oracle to recover all committed transactions up to the exact time a failure occurred. Operating in this mode enables tablespace and datafile backups.

The internal sequence numbers discussed earlier act as the guide for Oracle while it is using redo logs and archived redo logs to restore a database.

ARCHIVELOG mode and automatic archiving

Starting with Oracle Database 10*g*, automatic archiving for an Oracle Database is enabled with the following SQL command:

```
ARCHIVE LOG START
```

If the database is in ARCHIVELOG mode, Oracle marks the redo logs for archiving as it fills them. The full logfiles must be archived before they can be reused. The `ARCHIVE`

`LOG START` command will by default turn on automatic archiving and the archivers are started.

Prior to Oracle Database 10*g*, logfiles marked as ready for archiving did not mean they would be automatically archived. You also needed to set a parameter in the initialization file with the syntax:

```
LOG_ARCHIVE_START = TRUE
```

Setting this parameter started an Oracle process to copy a full redo log to the archive log destination.

The archive log destination and the format for the archived redo log names are specified using two additional parameters, `LOG_ARCHIVE_DEST` and `LOG_ARCHIVE_FORMAT`. A setting such as the following:

```
LOG_ARCHIVE_DEST = C:\ORANT\DATABASE\ARCHIVE
```

specifies the directory to which Oracle writes the archived redo logfiles, and

```
LOG_ARCHIVE_FORMAT = ORCL%t_%s_%r.arc
```

defines the format Oracle will use for the archived redo log filenames. In this case, the filenames will begin with `ORCL` and will end with `.arc`. The parameters for the format wildcards are:

`%t`
: Include thread number as part of the filename

`%s`
: Include log sequence number as part of the filename

`%r`
: Include resetlogs ID as part of the filename

If you want the archived redo log filenames to include the thread number, log sequence number, and resetlogs ID with the numbers zero-padded, capitalize the parameters and set:

```
LOG_ARCHIVE_FORMAT = "ORCL%T_%S_%R.arc"
```

Since the initialization file is read when an Oracle instance is started, changes to these parameters do not take effect until an instance is stopped and restarted. Remember, though, that turning on automatic archiving does not put the database in ARCHIVELOG mode. Similarly, placing the database in ARCHIVELOG mode does not enable the automatic archiving process.

You should also make sure that the archive log destination has enough room for the logs Oracle will automatically write to it. If the archive logfile destination is full, Oracle will hang since it can't archive additional redo logfiles.

Figure 2-7 illustrates redo log use with archiving enabled.

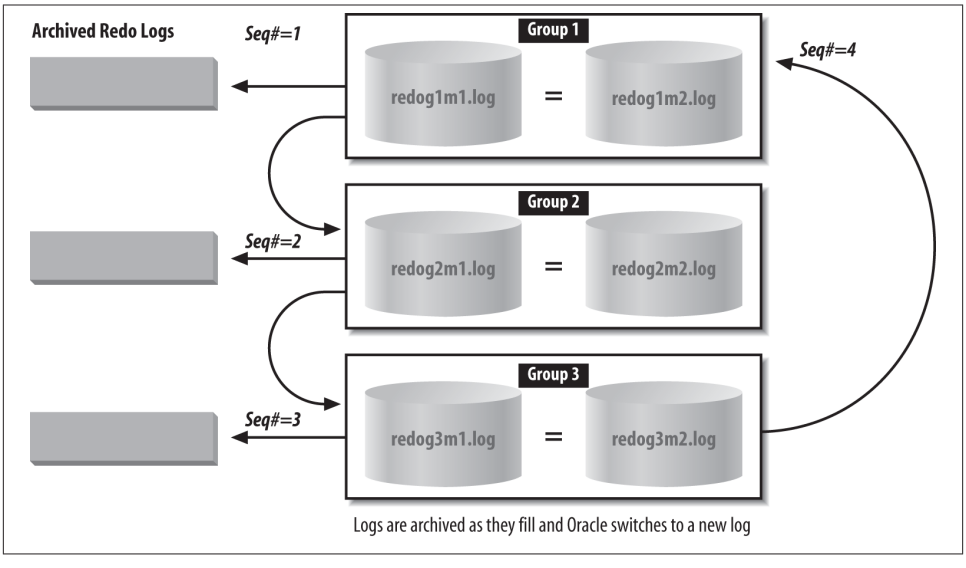


Figure 2-7. Cycling redo logs with archiving

The archived redo logs are critical for database recovery. Just as you can duplex the online redo logs, you can also specify multiple archive log destinations. Oracle will copy filled redo logs to specified destinations. You can also specify whether all copies must succeed or not. The initialization parameters for this functionality are as follows:

LOG_ARCHIVE_DUPLEX_DEST
: Specifies an additional location for redundant redo logs.

LOG_ARCHIVE_MIN_SUCCEED_DEST
: Indicates whether the redo log must be successfully written to one or all of the locations. Valid values are 1 through 10 if multiplexing and 1 or 2 if duplexing.

See your Oracle documentation for the additional parameters and views that enable and control this functionality.

Instance Memory and Processes

An Oracle instance can be defined as an area of shared memory and a collection of background processes. The area of shared memory for an instance is called the *System Global Area*, or SGA. The SGA is not really one large undifferentiated section of memory—it's made up of various components that we'll examine in the next section. All the processes of an instance—system processes and user processes—share the SGA. There is one SGA for an Oracle instance.

Prior to Oracle9*i*, the size of the SGA was set when the Oracle instance was started. The only way to change the size of the SGA or any of its components was to change the initialization parameter and then stop and restart the instance. Since Oracle9*i*, you can also change the size of the SGA or its components while the Oracle instance is running. Oracle9*i* also introduced the concept of the *granule*, which is the smallest amount of memory that you can add to or subtract from the SGA.

Oracle Database 10*g* introduced Automatic Shared Memory Management, while Oracle Database 11*g* added Automatic Memory Management for the SGA and PGA instance components. Whenever the MEMORY_TARGET (new to Oracle Database 11*g*) or SGA_TARGET initialization parameter is set, the database automatically distributes the memory among various SGA components providing optimal memory management. The shared memory components automatically sized include the shared pool (manually set using SHARED_POOL_SIZE), the large pool (LARGE_POOL_SIZE), the Java pool (JAVA_POOL_SIZE), the buffer cache (DB_CACHE_SIZE), and the streams pool (STREAMS_POOL_SIZE). Automatic memory management initialization parameters can be set through Oracle Enterprise Manager.

The background processes interact with the operating system and each other to manage the memory structures for the instance. These processes also manage the actual database on disk and perform general housekeeping for the instance.

Figure 2-8 illustrates the memory structures and background processes discussed in the following section.

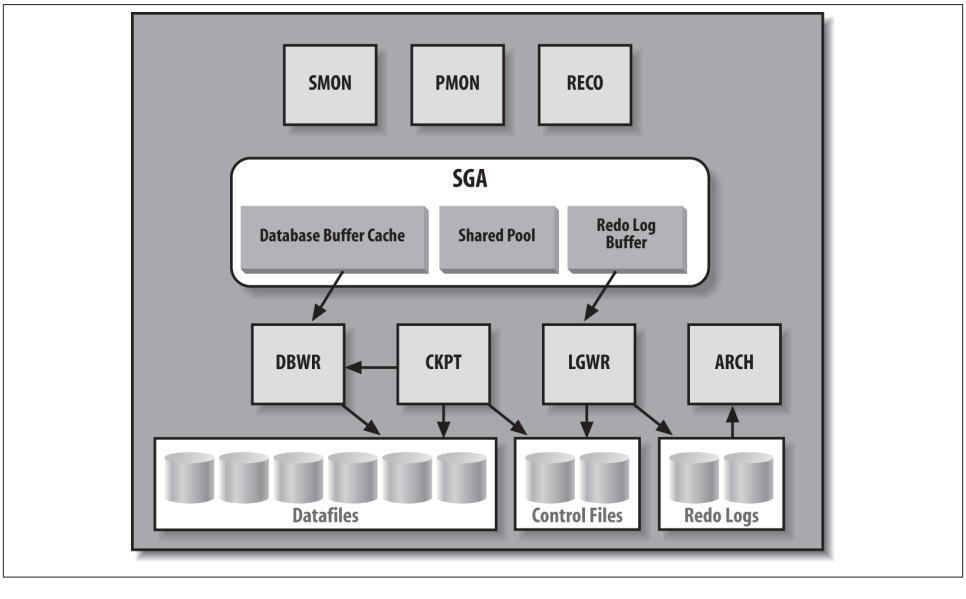


Figure 2-8. An Oracle instance

Additional background processes may exist when you use certain other features of the database; for example, shared servers (formerly the Multi-Threaded Server or MTS prior to Oracle9*i*), or job queues and replication.

Memory Structures for an Instance

As shown in Figure 2-8, the System Global Area is composed of multiple areas. These include a database buffer cache, a shared pool, and a redo log buffer as shown in the figure, and also possibly a Java pool, a large pool, and a Streams pool. The following sections describe these areas of the SGA. For a more detailed discussion of performance and the SGA, see "How Oracle Uses the System Global Area" on page 191 in Chapter 7.

Database buffer cache

The database buffer cache holds blocks of data retrieved from the database. This buffer between the users' requests and the actual datafiles improves the performance of the Oracle Database. If a piece of data can be found in the buffer cache (for example, as the result of a recent query), you can retrieve it from memory without the overhead of having to go to disk. Oracle manages the cache using a least recently used (LRU) algorithm. If a user requests data that has been recently used, the data is more likely to be in the database buffer cache; data in the cache can be delivered immediately without a disk-read operation being executed.

When a user wants to read a block that is not in the cache, the block must be read and loaded into the cache. When a user makes changes to a block, those changes are made to the block in the cache and a record of the change is written to the redo logfile. At some later time, those changes will be written to the datafile in which the block resides. This avoids making users wait while Oracle writes their changed blocks to disk.

This notion of waiting to perform I/O until absolutely necessary is common throughout Oracle. Disks are the slowest component of a computer system, so the less I/O performed, the faster the system runs. By deferring noncritical I/O operations instead of performing them immediately, an Oracle Database can deliver better performance.

The database buffer cache can be configured with buffer pools of the following types:

DEFAULT
: The standard Oracle Database buffer cache. All objects use this cache unless otherwise indicated.

KEEP
: For frequently used objects you wish to cache.

RECYCLE
: For objects you're less likely to access again.

Both the KEEP and RECYCLE buffer pools remove their objects from consideration by the LRU algorithm.

You can mark a table or index for caching in a specific buffer pool. This helps to keep more desirable objects in the cache and avoids the "churn" of all objects fighting for space in one central cache. Of course, to use these features properly you must be aware of the access patterns for the various objects used by your application.

Oracle Database 10*g* simplified management of buffer cache size by introducing a new dynamic parameter, DB_CACHE_SIZE. This parameter can be used to specify cache memory size and replaced the DB_BLOCK_BUFFERS parameter present in previous Oracle releases.

DB_CACHE_SIZE is automatically sized if MEMORY_TARGET or SGA_TARGET is set. Other initialization parameters include DB_KEEP_CACHE_SIZE and DB_RECYCLE_CACHE_SIZE and these must be manually sized if used.

Shared pool

The shared pool caches various constructs that can be shared among users. For example, SQL queries and query fragments issued by users and results are cached so they can be reused if the same statement is submitted again. PL/SQL functions are also loaded into the shared pool for execution and the functions and results are cached, again using an LRU algorithm. As of Oracle Database 11*g*, a PL/SQL function can be marked in such a way that its result will be cached to allow lookup rather than recalculation when it is called again using the same parameters. The shared pool is also used for caching information from the Oracle data dictionary, which is the metadata that describes the structure and content of the database itself.

You can specify a SHARED_POOL_SIZE initialization parameter, or it will be automatically sized if MEMORY_TARGET or SGA_TARGET is specified. Note that prior to Oracle Database 10*g*, "out of memory" errors were possible if the shared pool was undersized, but current Oracle Database releases now can leverage automatic shared memory tuning.

Redo log buffer

The redo log buffer caches redo information until it is written to the physical redo logfiles stored on a disk. This buffer also improves performance. Oracle caches the redo until it can be written to a disk at a more optimal time, which avoids the overhead of constantly writing the redo logs to disk.

Other pools in the SGA

The SGA includes several other pools:

Large pool
: Provides memory allocation for various I/O server processes, backup, and recovery, and provides session memory where shared servers and Oracle XA for transaction processing are used.

Java pool
: Provides memory allocation for Java objects and Java execution, including data in the Java Virtual Machine in the database.

Streams pool
: Provides memory allocation used to buffer Oracle Streams queued messages in the SGA instead of in database tables and provides memory for capture and apply. Note that Oracle GoldenGate is now the recommended solution, instead of Streams, for queuing messages to be delivered among databases.

Dynamic initialization parameters available for these pools include LARGE_POOL_SIZE, JAVA_POOL_SIZE, and STREAMS_POOL_SIZE. These are automatically set if MEMORY_TARGET or SGA_TARGET is specified.

Automatic PGA management

Oracle automatically manages the memory allocated to an instance Program Global Area (PGA). The PGA consists of session memory and a private SQL area. There is a PGA allocated for each service, which corresponds to a pluggable database in Oracle Database 12*c*. The memory amount can be controlled by setting the PGA_AGGREGATE_TARGET initialization parameter. Automatic PGA management, available since Oracle Database 10*g*, greatly simplified management of SQL work areas and eliminated the need to set several different initialization parameters that previously existed. As of Oracle Database 11*g*, PGA memory allocation is automatically tuned along with the SGA memory allocations by setting MEMORY_TARGET. With Oracle Database 12*c*, this control has been refined by adding a parameter for the PGA_AGGREGATE_LIMIT. This parameter sets a hard limit on the total amount of memory that the PGA can use; when this limit is reached, the sessions using the greatest amount of the PGA are paused until the memory usage drops.

Background Processes for an Instance

The most common background processes are shown in Figure 2-8 and vary from Oracle release to release. Among the background processes in Oracle Database 12*c* are the following:

Database Writer (DBWn)
: Writes database blocks from the database buffer cache in the SGA to the datafiles on disk. An Oracle instance can have up to 20 DBW processes to handle the I/O load to multiple datafiles—hence the notation DBW*n*. Most instances run one DBW. DBW writes blocks out of the cache for two main reasons:

  - If Oracle needs to perform a checkpoint (i.e., to update the blocks of the datafiles so that they "catch up" to the redo logs). Oracle writes the redo for a transaction when it's committed, and later writes the actual blocks. Periodically, Oracle

performs a checkpoint to bring the datafile contents in line with the redo that was written out for the committed transactions.

- If Oracle needs to read blocks requested by users into the cache and there is no free space in the buffer cache, the blocks written out are the least recently used blocks. Writing blocks in this order minimizes the performance impact of losing them from the buffer cache.

Log Writer (LGWR)
: Writes the redo information from the log buffer in the SGA to all copies of the current redo logfile on disk. As transactions proceed, the associated redo information is stored in the redo log buffer in the SGA. When a transaction is committed, Oracle makes the redo information permanent by invoking the Log Writer to write it to disk.

System Monitor (SMON)
: Maintains overall health and safety for an Oracle instance. SMON performs crash recovery when the instance is started after a failure and coordinates and performs recovery for a failed instance when you have more than one instance accessing the same database, as with Real Application Clusters. SMON also cleans up adjacent pieces of free space in the datafiles by merging them into one piece and gets rid of space used for sorting rows when that space is no longer needed.

Process Monitor (PMON)
: Watches over the user processes that access the database. If a user process terminates abnormally, PMON is responsible for cleaning up any of the resources left behind (such as memory) and for releasing any locks held by the failed process.

Archiver (ARCn)
: Reads the redo logfiles once Oracle has filled them and writes a copy of the used redo logfiles to the specified archive log destination(s).

  Up to 10 Archiver processes are possible—hence the notation ARC*n*. LGWR will start additional Archivers as needed, based on the load, up to a limit specified by the initialization parameter LOG_ARCHIVE_MAX_PROCESSES. By default, this initialization parameter has a value of 2 and is rarely changed.

Checkpoint (CKPT)
: Updates datafile headers whenever a checkpoint is performed.

Recover (RECO)
: Automatically cleans up failed or suspended distributed transactions.

Dispatcher
: Optional background processes used when shared server configurations are deployed.

Global Cache Service (LMS)
: Manages resources for Real Application Clusters and inter-instance resource control.

Job Queue
: Provides a scheduler service used to schedule user PL/SQL statements or procedures in batch.

Queue Monitor (QMNn)
: Monitors Oracle Streams message queues with up to 10 monitoring processes supported.

Automatic Storage Management (ASM) processes
: RBAL coordinates rebalancing of activities for disk groups. ORB*n* performs the actual rebalancing. ASMB provides communication between the database and the ASM instance.

Configuration, Engineered Systems, and the Cloud

The previous pages have introduced you to the basics of Oracle Database architecture. If you are brand new to Oracle, it may seem like a fair amount to digest. And, indeed, creating a database that can manage data state integrity reliably for tens of thousands of users is a nontrivial task that requires a fair amount of underlying architecture.

One of the constant complaints about the Oracle Database was that there were too many dials to twist and too much maintenance and configuration to attend to. The good part about these configuration options is that they allow fine-tuning of the general purpose Oracle Database to perform and scale optimally for any purpose you choose for your particular requirements. However, as we mentioned earlier, much of the initial setup and tuning is no longer required, as the database initialization parameters, with few exceptions, are set to defaults and the database has become more self-tuning and self-managing with each subsequent release.

There is a downside to exposing this flexibility. Every option that can be set can also be set incorrectly. Even highly knowledgeable database administrators may occasionally suffer from this fault, either through changing circumstances or lack of understanding. In order to reduce this downside, Oracle Corporation has been focusing on two areas: engineered systems and the cloud.

Engineered systems, such as Exadata, are, among other things, pre-assembled and pre-configured systems. The pre-assembly makes it faster to start using your Oracle Database, while the pre-configuration means that the system will be properly configured and balanced to operate optimally without the need for extensive tuning that was often required in the past. Oracle has found that these systems frequently end up being more suitable and less problematic for customer needs than customized configurations.

Of course, if you want to get away from both configuration and all other ongoing maintenance tasks, you may want to consider the Oracle Database Cloud, a public cloud offering described in more detail in Chapter 15.

The Data Dictionary

Each Oracle Database includes a set of *metadata* that describes the data structure including table definitions and integrity constraints. The tables and views that hold this metadata are referred to as the Oracle *data dictionary*. All of the components discussed in this chapter have corresponding system tables and views in the data dictionary that fully describe the characteristics of the component. You can query these tables and views using standard SQL statements. Table 2-1 shows where you can find some of the information available about each of the components in the data dictionary.

The SYSTEM tablespace always contains the data dictionary tables. Data dictionary tables that are preceded by the V$ or GV$ prefixes are dynamic tables, which are continually updated to reflect the current state of the Oracle Database. Static data dictionary tables can have a prefix such as DBA_, ALL_, or USER_ to indicate the scope of the objects listed in the table.

With the introduction of the multitenant architecture in Oracle 12*c*, a new level has been introduced to the Oracle Database dictionary. DBA_, ALL_ and USER_ views exist within the context of a pluggable database, while a new set of views with the prefix CDB_ are available in the root container, which aggregates the DBA_ views from all pluggable databases associated with that container. Common users defined within the container database will be able to see information for all PDBs for which they have privileges when querying CDB_ views. Users within a PDB will see data dictionary views just as if they were users within a standard non-CDB instance.

Table 2-1. Partial list of database components and their related data dictionary views

Component	Data dictionary tables and views
Database	V$DATABASE, V$VERSION, V$INSTANCE
Pluggable databases	V$CONTAINERS, V$PDBS
Shared server	V$QUEUE, V$DISPATCHER, V$SHARED_SERVER
Connection pooling	DBA_CPOOL_INFO, V$CPOOL_STATS, V$CPOOL_CC_STATS
Tablespaces	USER_FREE_SPACE, DBA_FREE_SPACE, V$TEMPFILE, DBA_USERS, DBA_TS_QUOTAS
Control files	V$CONTROLFILE, V$PARAMETER, V$CONTROLFILE_RECORD_SECTION
Datafiles	V$DATAFILE, V$DATAFILE_HEADER, DBA_DATA_FILES, DBA_EXTENTS, USER_EXTENTS
Segments	DBA_SEGMENTS, USER_SEGMENTS
Extents	DBA_EXTENTS, USER_EXTENTS
Redo logs	V$THREAD, V$LOG, V$LOGFILE, V$LOG_HISTORY
Undo	V$UNDOSTAT, V$ROLLSTAT, V$TRANSACTION

Component	Data dictionary tables and views
Archiving status	V$DATABASE, V$LOG, V$ARCHIVED_LOG, V$ARCHIVE_DEST
Database instance	V$INSTANCE, V$PARAMETER, V$SYSTEM_PARAMETER
Memory structure	VSGA, VSGASTAT, V$SGAINFO, V$SGA_DYNAMIC_COMPONENTS, V $SGA_DYNAMIC_FREE_MEMORY, VSGA_RESIZE_OPS, VSGA_RESIZE_CURRENT_OPS, V $MEMORY_TARGET_ADVICE, V$SGA_TARGET_ADVICE, V$PGA_TARGET_ADVICE
Work area memory	V$PGASTAT, V$SYSSTAT
Processes	V$PROCESS, V$BGPROCESS, V$SESSION
Alerting	DBA_THRESHOLDS, DBA_OUTSTANDING_ALERTS, DBA_ALERT_HISTORY, V$ALERT_TYPES, V$METRIC
Performance monitoring	V$LOCK, DBA_LOCK, V$SESSION_WAIT, V$SQLAREA, V$LATCH
RMAN recovery	V$RECOVER_FILE
User passwords	V$PWFILE_USERS
Tables	DBA_TABLES, ALL_TABLES, USER_TABLES
Indexes	DBA_INDEXES, ALL_INDEXES, USER_INDEXES
Data dictionary	DBA_OBJECTS, ALL_OBJECTS, USER_OBJECTS

CHAPTER 3

Installing and Running Oracle

If you've been reading this book sequentially, you should understand the basics of the Oracle Database architecture by now. This chapter begins with a description of how to install a database and get it up and running. (If you've already installed your Oracle Database software, you can skim through this first section.) We'll describe how to create an actual database and how to configure the network software needed to run Oracle, with a brief detour to look at how cloud computing changes this initial process. Finally, we'll discuss how users access databases and begin a discussion of how to manage databases—a topic that will be continued in subsequent chapters.

Installing Oracle

Prior to Oracle8*i*, the Oracle installer came in both character and GUI versions for Unix. The Unix GUI ran in Motif using the X Window system. Windows NT came with a GUI version only. Since Oracle8*i*, the installer has been Java-based.

The Oracle installer is one of the first places in which you can see the benefits of the portability of Java; the installer looks and functions the same way across all operating systems. For some time now, installing Oracle has been fairly simple, requiring only a few mouse clicks and answers to some questions about options and features.

Oracle made great strides in further simplifying installation with Oracle Database 10*g*. Both that install and the installation of Oracle Database 12*c* can be accomplished in less than 20 minutes.

The current version of the Oracle Universal Installer begins the process by checking the target environment to make sure there are enough resources for the Oracle Database. If the target is a bit light, you will be informed with a warning and given the option to continue.

As part of the installation process, the Installer also runs the Net Configuration Assistant and the Database Configuration Assistant so that you will end up with a working Oracle instance when the process is complete.

If, for some reason, the installation fails, the commands that did not succeed are listed in a logfile, which helps you understand where the problem may lie and gives you a handy set of commands you can run yourself once the problem is fixed.

Although the installation process is now the same for all platforms, there are still particulars about the installation of Oracle that relate to specific platforms. Each release of the Oracle Database Server software is shipped with its own set of documentation. Included in each release is an installation guide, release notes (which include installation information added after the installation guide was published), and a "getting started" book. You should read all of these documents prior to starting the installation process, since each of them contains invaluable information about the specifics of the installation. You will need to consider details such as where to establish the Oracle Home directory and where database files will reside. These issues are covered in detail in the documentation. Online documentation is shipped on the database server media, and this provides additional information regarding the database and related products.

You'll typically find the installation guide in the server software case. The installation guide includes system requirements (memory and disk), pre-installation tasks, directions for running the installation, and notes regarding migration of earlier Oracle Databases to the current release. You should remember that complete installation of the software includes not only loading the software, but also configuring and starting key services.

One of the more important decisions you needed to make before actually installing Oracle in older releases concerned the directory structure and naming conventions you would follow for the files that make up a database. Clear, consistent, and well-planned conventions were crucial for minimizing human errors in system and database administration. Today, this naming is largely automated during the installation process. Some of the more important database naming that takes place includes the following:

- Disk or mount point names
- Directory structures for Oracle software and database files
- Database filenames: control files, database files, and redo logfiles

The Optimal Flexible Architecture (OFA), described in the next section, became the basis for naming conventions for all of these files. In engineered systems, such as the Exadata Database Machine, an initial Oracle Database is installed as part of Oracle's Start-Up Pack services with naming conventions based on OFA.

Optimal Flexible Architecture

Oracle consultants working at large Oracle sites created (out of necessity) a comprehensive set of standards for database directory structures and filenames prior to Oracle's introduction of more automated installation procedures. This set of standards is called *An Optimal Flexible Architecture for a Growing Oracle Database* or, as it is lovingly known in the Oracle community, the OFA. For example, the OFA provides a clear set of standards for handling multiple databases and multiple versions of Oracle if deployed on the same machine. It includes recommendations for mount points, directory structures, filenames, and scripting techniques. Anyone who knows the OFA can navigate an Oracle environment to quickly find the software and files used for the database and the instance. This standardization increased productivity and avoided errors.

The OFA standards are embedded in the Oracle installer. System administrators and database administrators working with Oracle will find understanding the OFA worthwhile, even if your Oracle system is already installed. OFA documentation is included in the Oracle installation guide.

Supporting Multiple Oracle Versions on a Machine

You can install and run multiple versions of Oracle on a single-server machine. All Oracle products use a directory referred to by the environment or system variable ORACLE_HOME to find the base directory for the software they will use. Because of this, you can run multiple versions of Oracle software on the same server, each with a different ORACLE_HOME variable defined. Whenever a piece of software accesses a particular version of Oracle, the software simply uses the proper setting for the ORACLE_HOME environment variable.

Oracle supports multiple ORACLE_HOME variables on Unix and Windows systems by using different directories. The OFA provides clear and excellent standards for this type of implementation.

Upgrading an Oracle Database

Oracle Database 10*g* added two additional features that apply to upgrading an existing Oracle Database: the Database Upgrade Assistant and support for rolling upgrades.

If you want to upgrade a single instance, you can use the Database Upgrade Assistant, which can be started from the Oracle Universal Installer. As of Oracle Database 11*g*, you can upgrade from the free version of Oracle, Oracle XE, to a single instance of another edition with the Database Upgrade Assistant.

One of the longstanding problems with upgrades has been the requirement to bring down the database, upgrade the database software, and then restart the database. This necessary downtime can impinge on your operational requirements. If you are using a

Real Application Clusters implementation since Oracle Database 10*g*, you can perform a *rolling upgrade*. A rolling upgrade allows you to bring down some of the nodes of the cluster, upgrade their software, and then bring them back online as part of the cluster. You can then repeat this procedure with the other nodes. The end result is that you can achieve a complete upgrade of your Oracle Database software without having to bring down the database.

Creating a Database

As we noted in Chapter 2, Oracle can be used to support a variety of workloads. You should take a two-step approach for any new databases you create. First, understand the purpose of the database, and then create the database with the appropriate parameters.

Planning the Database

You should spend some time learning the purpose of an Oracle Database before you create the database itself. Consider what the database will be used for and how much data it will contain. You should understand the underlying hardware that you'll use—the number and type of CPUs, the amount of memory, the number of disks, the controllers for the disks, and so on. Because the database is stored on the disks, many tuning problems can be avoided with proper capacity and I/O subsystem planning.

Planning your database and the supporting hardware requires insights into the scale or size of the workload and the type of work the system will perform. Some of the considerations that will affect your database design and hardware configuration include the following:

How many users will the database have?
: How many users will connect simultaneously and how many will concurrently perform transactions or execute queries?

Is the database supporting OLTP applications or data warehousing?
: This distinction leads to different types and volumes of activity on the database server. For example, online transaction processing (OLTP) systems usually have a larger number of users performing smaller transactions, including a significant percentage of write operations, while data warehouses usually have a smaller number of users performing larger queries.

What is the expected size and number of database objects?
: How large will these objects be initially and what growth rates do you expect?

What are the access patterns for the various database objects?
: Some objects will be more popular than others. Understanding the volume and type of activity in the database is critical to planning and tuning your database. Some

people employ a so-called *CRUD matrix* that contains Create, Read, Update, and Delete indicators, or even estimates for how many operations will be performed for each key object used by a business transaction. These estimates may be per minute, per hour, per day, or for whatever time period makes sense in the context of your system. For example, the CRUD matrix for a simple employee update transaction might be as shown in Table 3-1, with the checkmarks indicating that each transaction performs the operation against the object shown.

Table 3-1. Access patterns for database objects

Object	Create	Read	Update	Delete
EMP	✓	✓		
DEPT		✓		
SALARY		✓	✓	

How much hardware do I have now, and how much will I add as the database grows?
Disk drives tend to get cheaper and cheaper. Suppose you're planning a database of 10 TB that you expect to grow to 90 TB over the next two years. You may have all the disk space available to plan for the 90 TB target, but it's more likely that you'll buy a smaller amount to get started and add disks as the database grows. It's important that you plan the initial layout with the expected growth in mind, although Real Application Clusters make it much easier to scale out your database easily.

Prior to Oracle9*i*, running out of tablespace in the middle of a batch operation meant that the entire operation had to be rolled back. Oracle9*i* introduced the concept of *resumable space allocation*. When an operation encounters an out-of-space condition, if the resumable statement option has been enabled for the session, the operation is suspended for a specific length of time, which allows the operator to correct the out-of-space condition. You even have the option to create an AFTER SUSPEND trigger to fire when an operation has been suspended.

With Automatic Storage Management (ASM), introduced in Oracle Database 10*g*, you can add additional disk space or take away disks without interrupting database service. Although you should still carefully estimate storage requirements, the penalty for an incorrect judgment, in terms of database downtime, is significantly reduced with ASM.

What are the availability requirements?
What elements of redundancy, such as additional disk drives, do you need to provide the required availability? ASM also provides automatic mirroring for data with different redundancies available, which can help to provide data resiliency.

What are my performance requirements?
: What response times do your users expect, and how much of that time can you give them? Will you measure performance in terms of average response time, maximum response time, response time at peak load, total throughput, or average load?

What are my security requirements?
: Will the application, the operating system, or the Oracle Database (or some combination of these) enforce security? Do I need to implement this security in my design or with additional security options for the Oracle Database? These options are described in Chapter 6.

The Value of Estimating

Even if you are unsure of things such as sizing and usage details, take your best guess as to initial values and growth rates and document these estimates. As the database evolves, you can compare your initial estimates with emerging information to react and plan more effectively. For example, suppose you estimate that a certain table will be 500 GB in size initially and will grow at 300 GB per year, but when you are up and running you discover that the table is actually 300 GB, and six months into production you discover that it has grown to 800 GB. You can now revise your plans to reflect the higher growth rate and thereby avoid space problems. Comparing production measures of database size, growth, and usage patterns with your initial estimates will provide valuable insights to help you avoid problems as you move forward. In this way, documented guesses at an early stage are useful later on.

The same is true for key requirements such as availability and performance. If the exact requirements are not clear, make some assumptions and document them. These core requirements will heavily influence the decisions you make regarding redundancy and capacity. As the system evolves and these requirements become clearer, the history of these key decision criteria will be crucial in understanding the choices that you made and will make in the future.

The Automatic Workload Repository (AWR), first available in Oracle Database 10*g*, maintains a history of workload and performance measurements, which are used by the Automatic Database Diagnostic Monitor (ADDM) to spot performance anomalies. You can also use AWR to track ongoing changes in workload.

Tools for Creating Databases

Traditionally, there have been two basic ways to create an Oracle Database:

- Use the graphical installer tool
- Run character-mode scripts

These days, the Oracle Database calls the Database Configuration Assistant (DBCA) Installer to create and configure the Oracle Database and its associated software, such as the Oracle Listener (described below). It is written in Java and therefore provides the same look and feel across platforms. The Installer is a quick and easy way to create, modify, or delete a database. It allows you to create a typical pre-configured database (with minimal input required) or a custom database (which involves making some choices and answering additional questions). The Installer in the midst of an installation is shown in Figure 3-1.

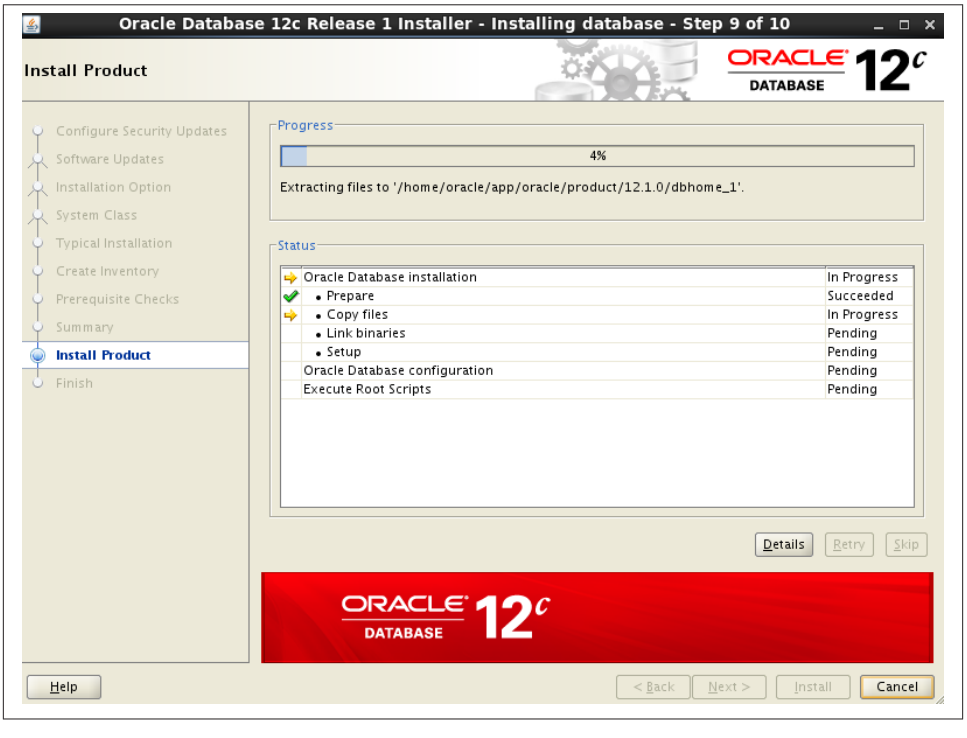


Figure 3-1. Oracle Database 12c Installer

The DBCA walks you through the process of creating a database, including checking for prerequisites, copying files and linking binaries, and setting up your network configuration.The DBCA has become more intelligent over time, and the 12*c* version of this software will even generate a script file that you can simply run to correct any problems with prerequisites in the system software.The entire installation process takes less than 20 minutes for a standard desktop configuration.

The alternative method for creating a database is to create or edit an existing SQL script that executes the various required commands. Most Oracle DBAs have a preferred script that they edit as needed. In Oracle7 and Oracle8, you executed the script using a

character-mode utility called Server Manager; since Oracle8*i*, you could use SQL*Plus. The Oracle software installation media for earlier versions of the Oracle Database also includes a sample script called *BUILD_DB.SQL*, described in the Oracle documentation. Today, most users choose to create the database with the standard installer interface; the command line method is actively discouraged for Oracle Database 12*c* Databases.

Oracle Net Services and Oracle Net

The overwhelming majority of applications that use an Oracle Database run on a different machine and connect to the Oracle Database over a network. The software used to transparently implement this connection process is known as Oracle Net Services.

Oracle Net Services handle interaction with underlying network protocols, such as TCP/IP, and the coordination between client requests and the server processes that fulfill them. Oracle Net Services include technology for clients to discover the appropriate database instance, establish communication with that instance, and maintain an ongoing interaction between the client and the server.

Oracle Net is a layer of software that allows different physical machines to communicate for the purpose of accessing an Oracle Database.

The term "Oracle Net Services" in Oracle refers to all the components of Oracle Net, including dispatchers, listeners, and shared servers; these are explained later in this chapter.

A version of Oracle Net runs on the client machine and on the database server, and allows clients and servers to communicate over a network using virtually any popular network protocol. Oracle Net can also perform network protocol interchanges. For example, it allows clients that are speaking LU 6.2 to interact with database servers that are speaking TCP/IP.

Oracle Net also provides *location transparency*—that is, the client application does not need to know the server's physical location. The Oracle Net layer handles the communications, which means that you can move the database to another machine and simply update the Oracle Net configuration details accordingly. The client applications will still be able to reach the database, and no application changes will be required.

Oracle Net supports the notion of *service names*, or *aliases*. Clients provide a service name or Oracle Net alias to specify which database they want to reach without having to identify the actual machine or instance for the database. Oracle Net looks up the actual machine and the Oracle instance, using the provided service name, and transparently routes the client to the appropriate database.

Resolving Oracle Net Service Names

With older versions of Oracle Net, you could use any of the following options to resolve the service name the client specifies into the host and instance names needed to reach an Oracle Database:

Local name resolution
: For local name resolution, you install a file called *TNSNAMES.ORA* on each client machine that contains entries that provide the host and Oracle instance for each Oracle Net alias. You must maintain this file on the client machines if any changes are made to the underlying database locations. Your network topology is almost certain to change over time, so use of this option can lead to an increased maintenance load. If you are using Oracle Internet Directory or other centralized directories, described later in this section, you do not need a *TNSNAMES.ORA* file. This method is known as local naming.

Oracle Names service
: Oracle Names was supported in earlier Oracle releases, providing a way to eliminate the need for a *TNSNAMES.ORA* file on each client. That was the good part. The bad part was that Oracle Names was a proprietary solution. Since Oracle Internet Directory is based on standards and provides this functionality, Oracle declared Oracle Names obsolete after the Oracle9*i* release.

Oracle Internet Directory or other LDAP directories
: The need for a centralized naming service extends far beyond the Oracle environment. In fact, there is a well-defined standard for accessing this type of information, the Lightweight Directory Access Protocol (LDAP). As of the Oracle Database 11*g* release, Oracle Internet Directory (OID) is a part of Fusion. OID is an LDAP-enabled directory that can fulfill the same role as the previously available Oracle Names service. Since Oracle Database 10*g*, you can export directory entries to create a local *TNSNAMES.ORA* file; this file may be used for clients not using the directory or if the directory is unavailable. This method is known as directory naming.

Host naming
: Clients can simply use the name of the host on which the instance runs. This is valid for TCP/IP networks with a mechanism in place for resolving the hostname into an IP address. For example, the Domain Name Service (DNS) translates a hostname into an IP address, much as Oracle Names translates service names. Since Oracle Database 10*g*, you can use this method with either a host name, domain-qualified if appropriate, or a TCP/IP address, but the connection will not support advanced services such as connection pooling. This method is known as Easy Connect.

Third-party naming services
: Oracle Net can interface with external or third-party naming and authentication services such as Kerberos or RADIUS. Use of such services may require Oracle Advanced Security.This method is known as external naming.

These name resolution options are not mutually exclusive. For example, you can use Oracle Internet Directory and local name resolution (*TNSNAMES.ORA* files) together. In this case, you specify the order Oracle should use in resolving names in the *SQLNET.ORA* file (for example, check OID first, and if the service name isn't resolved, check the local *TNSNAMES.ORA* file). This is useful for cases in which there are corporate database services specific to certain clients. You would use OID for the standard corporate database services, such as email, and then use *TNSNAMES.ORA* entries for the client-specific database services, such as a particular development database.

You also have the option to connect directly to an Oracle Database with what Oracle refers to as the *easy connect naming method*. This method uses the host name or TCP/IP identifier for the Oracle server machine and the name of the Oracle Database instance. The method is limited to use with TCP/IP networks, and is recommended only for fairly small installations where the host identifier is rarely changed.

From Oracle Database 11*g* on, Oracle Databases have used the concept of *server registration*. An instance is configured to be associated with one or more services. When the instance starts up, the instance dynamically informs the listeners of its associated services. Since this registration is dynamic, the listeners can also keep track of the availability of various instances associated with a service. An individual service can be associated with more than one database instance, and a single database instance can support more than one server.

Global Data Services

Oracle Database 12*c* introduces the concept of Global Data Services. Global Data Services give you the ability to have multiple database instances, even instances that are geographically dispersed, participate in a single global service. Oracle provides an infrastructure to allow for name resolution, and additional services, such as GoldenGate and DataGuard, are used to synchronize the information in different instances participating in a global service.

Oracle Net Manager

The Oracle Net Manager is written in Java, provides the same look and feel across platforms, and is typically first accessed from the Installer. The Oracle Net configuration files have a very specific syntax with multiple levels of nested brackets. Using the Oracle Net Manager allows you to avoid the errors that are common to handcoded files.

Oracle Enterprise Manager Cloud Control also gives you the ability to manage network configurations from the Enterprise Manager environment.

Debugging Network Problems

If you're having a problem with your network, one of the first steps toward debugging the problem is to check that the Oracle Net files were generated, not handcoded. If you're in doubt, back up the current configuration files and use the Oracle Net Manager to regenerate them. In fact, when Oracle Support assists customers with Oracle Net problems, one of the first questions they ask is whether or not the files were handcoded.

Oracle Connection Pooling

Many Oracle-based systems use a middle-tier server. This server is responsible for a number of tasks, including accepting requests from clients and passing them on to the database server. Since many clients only need a connection for a brief period, the middle tier can perform *connection pooling*. Connection pooling allows the middle tier to establish and manage a pool of connections that are shared between multiple clients, reducing the overall connection overhead needed by the database instance. The Oracle Connection Manager is responsible for the tasks of managing a connection pool from a middle tier.

Oracle Database 12*c* also supports *database resident connection pooling*. This capability allows connection pooling which is implemented in the database server without the need for a middle-tier component, so systems that require or desire a direct connection to the database can still benefit from the reduced overhead of shared connections.

Auto-Discovery and Agents

Beginning with Oracle 7.3, Oracle provided auto-discovery features that allowed it to find new databases automatically. Support for auto-discovery increased and improved with each Oracle release since then. Since Oracle8*i*, the Universal Installer and Oracle Net Manager work together smoothly to automatically configure your Oracle Net network.

A key piece of the Oracle network that enables auto-discovery is the Oracle Intelligent Agent. The agent is a piece of software that runs on the machine with your Oracle Database(s). It acts as an agent for other functions that need to find and work with the database on the machine. For example, the agent knows about the various Oracle instances on the machine and handles critical management functions, such as monitoring the database for certain events and executing jobs. The agent provides a central point for auto-discovery: Oracle Net discovers instances and databases by interrogating the

agent. We'll examine the general use of agents and their role in managing Oracle in Chapter 5.

Oracle Net Configuration Files

Oracle Net requires several configuration files. The default location for the files used to configure an Oracle Net network are as follows:

- On Windows servers, *ORACLE_HOME\network\admin* for Oracle8*i* and more current releases
- On Unix servers, *ORACLE_HOME/network/admin*

You can place these files in another location, in which case you must set an environment or system variable called TNS_ADMIN to the nondefault location. Oracle then uses TNS_ADMIN to locate the files. The vast majority of systems are configured using the default location.

The files that form a simple Oracle Net configuration used for local naming are as follows:

LISTENER.ORA
: Contains details for configuring the Oracle Net Listener, such as which instances or services the Listener is servicing. As the name implies, the Listener "listens" for incoming connection requests from clients that want to access the Oracle Database over the network. For details about the mechanics of the Listener's function, see the later section "Oracle Net and Establishing Network Connections" on page 81. Since Oracle Database 11*g*, services can dynamically register with listeners, as described above.

TNSNAMES.ORA
: Decodes a service name into a specific machine address and Oracle instance for the connection request. (If you're using OID or another directory service, as described earlier, you don't need to use the *TNSNAMES.ORA* file as part of your configuration.) This file is key to Oracle Net's location transparency. If you move a database from one machine to another, you can simply update the *TNSNAMES.ORA* files on the various clients to reflect the new machine address for the existing service name. For example, suppose that clients reach the database using a service name of SALES. The *TNSNAMES.ORA* file has an entry for the service name SALES that decodes to a machine named HOST1 and an Oracle instance called PROD. If the Oracle Database used for the SALES application is moved to a machine called HOST2, the *TNSNAMES.ORA* entry is updated to use the machine name HOST2. Once the *TNSNAMES.ORA* files are updated, client connection requests will be routed transparently to the new machine with no application changes required.

SQLNET.ORA
: Provides important defaults and miscellaneous configuration details. For example, *SQLNET.ORA* contains the default domain name for your network.

For directory naming, you can either use an *LDAP.ORA* configuration file, described below, or use a standard DNS server to locate the LDAP server:

LDAP.ORA
: For Oracle8*i* and later releases, the *LDAP.ORA* file contains the configuration information needed to use an LDAP directory, such as the Oracle Internet Directory. This information includes the location of the LDAP directory server and the default administrative context for the server. This is no longer required for an LDAP server that is registered with the Domain Name Server (DNS) since Oracle Database 10*g*.

As mentioned in Chapter 2, Oracle9*i* added a server parameter file, named *SPFILE*, which provides storage for system parameters you have changed while your Oracle9*i* instance is running, using the `ALTER SYSTEM` command. With the *SPFILE*, these new parameter values are preserved and used the next time you restart your Oracle instance. You can indicate whether a particular change to a system parameter is intended to be persistent (in which case it will be stored in the *SPFILE*) or temporary.

The *SPFILE* is a binary file that is kept on the server machine. By default, an Oracle9*i* or later instance will look for the *SPFILE* at startup and then for an instance of the *INIT.ORA* file.

The *SPFILE* can also be kept on a shared disk, so that it can be used to initialize multiple instances in an Oracle Real Application Clusters configuration.

Starting Up the Database

Starting a database is quite simple—on Windows you simply start the Oracle services (or specify that the services are started when the machine boots), and on Unix and Linux you issue the `STARTUP` command from SQL*Plus, or through Enterprise Manager. While starting a database appears to be a single action, it involves an instance and a database and occurs in several distinct phases. When you start a database, the following actions are automatically executed:

1. *Starting the instance*. Oracle reads the instance initialization parameters from the *SPFILE* or *INIT.ORA* file on the server. Oracle then allocates memory for the System Global Area and starts the background processes of the instance. At this point, none of the physical files in the database have been opened, and the instance is in the NOMOUNT state. (Note that the number of parameters that must be defined in the *SPFILE* in Oracle Database 10*g* and Oracle Database 11*g* as part of the initial installation setup have been greatly reduced. We described the initialization parameters required in Oracle Database 12*c* in Chapter 2.)

There are problems that can prevent an instance from starting. For example, there may be errors in the initialization file, or the operating system may not be able to allocate the requested amount of shared memory for the SGA. You also need the special privilege SYSOPER or SYSDBA, granted through either the operating system or a password file, to start an instance.

2. *Mounting the database*. The instance opens the database's control files. The initialization parameter CONTROL_FILES tells the instance where to find these control files. At this point, only the control files are open. This is called the MOUNT state, and the database is accessible only to the database administrator. In this state, the DBA can perform limited types of database administration. For example, the DBA may have moved or renamed one of the database files. The datafiles are listed in the control file but aren't open in the MOUNT state. The DBA can issue a command (`ALTER DATABASE`) to rename a datafile. This command will update the control file with the new datafile name.
3. *Opening the database*. The instance opens the redo logfiles and datafiles using the information in the control file. At this point, the database is fully open and available for user access.

Shutting Down the Database

Logically enough, the process of shutting down a database or making it inaccessible involves steps that reverse those discussed in the previous section:

1. *Closing the database*. Oracle flushes any modified database blocks that haven't yet been written to the disk from the SGA cache to the datafiles. Oracle also writes out any relevant redo information remaining in the redo log buffer. Oracle then checkpoints the datafiles, marking the datafile headers as "current" as of the time the database was closed, and closes the datafiles and redologfiles. At this point, users can no longer access the database.
2. *Dismounting the database*. The Oracle instance dismounts the database. Oracle updates the relevant entries in the control files to record a clean shutdown and then closes them. At this point, the entire database is closed; only the instance remains.
3. *Shutting down the instance*. The Oracle software stops the background processes of the instance and frees, or deallocates, the shared memory used for the SGA.

In some cases (e.g., if there is a machine failure or the DBA aborts the instance), the database may not be closed cleanly. If this happens, Oracle doesn't have a chance to write the modified database blocks from the SGA to the datafiles. When Oracle is started again, the instance will detect that a crash occurred and will use the redo logs to automatically perform what is called *crash recovery*. Crash recovery guarantees that the changes for all committed transactions are done and that all uncommitted or in-flight

transactions will be cleaned up. The uncommitted transactions are determined after the redo log is applied and automatically rolled back.

Oracle Database 12*c* includes a new feature called Transaction Guard, which allows applications to determine the state of transactions that have been sent to the database server but not acknowledged. You can learn more about Transaction Guard in Chapter 9.

Accessing a Database

The previous sections described the process of starting up and shutting down a database. But the database is only part of a complete system—you also need a client process to access the database, even if that process is on the same physical machine as the database.

Server Processes and Clients

To access a database, a user connects to the instance that provides access to the desired database. A program that accesses a database is really composed of two distinct pieces —a client program and a server process—that connect to the Oracle instance. For example, running the Oracle character-mode utility SQL*Plus involves two processes:

- The SQL*Plus process itself, acting as the client
- The Oracle server process, sometimes referred to as a *shadow process*, that provides the connection to the Oracle instance

Server process

The Oracle server process always runs on the computer on which the instance is running. The server process attaches to the shared memory used for the SGA and can read from it and write to it.

As the name implies, the server process works for the client process—it reads and passes back the requested data, accepts and makes changes on behalf of the client, and so on. For example, when a client wants to read a row of data stored in a particular database block, the server process identifies the desired block and either retrieves it from the database buffer cache or reads it from the correct datafile and loads it into the database buffer cache. Then, if the user requests changes, the server process modifies the block in the cache and generates and stores the necessary redo information in the redo log buffer in the SGA. The server process, however, does not write the redo information from the log buffer to the redo logfiles, and it does not write the modified database block from the buffer cache to the datafile. These actions are performed by the Log Writer (LGWR) and Database Writer (DBWR) processes, respectively.

Client process

The client process can run on the same machine as the instance or on a separate computer. A network connects the two computers and provides a way for the two processes to talk to each other. In either case, the concept is essentially the same—two processes are involved in the interaction between a client and the database. When both processes are on the same machine, Oracle uses local communications via Inter Process Communication (IPC); when the client is on one machine and the database server is on another, Oracle uses Oracle Net over the network to communicate between the two machines.

Application Servers and Web Servers As Clients

Although the discussion in the previous section used the terms *client* and *server* extensively, please don't assume that Oracle is strictly a client/server database. Oracle was one of the early pioneers of client/server computing based on the notion of two tasks: a client and a server. But when you consider multitier computing involving web and application servers, the notion of a client changes somewhat. The "client" process becomes the middle tier, or application server.

You can logically consider any process that connects to an Oracle instance a client in the sense that it is served by the database. Don't confuse this usage of the term client with the actual client in a multitier configuration. The eventual client in a multitier model is some type of program providing a user interface—for example, a browser running an application composed of HTML and Javascript.

The Oracle WebLogic Server application server, which is part of the overall Oracle platform, is designed to act as this middle tier. WebLogic Server works seamlessly with the Oracle Database.

Figure 3-2 illustrates users connecting to an Oracle instance to access a database in both two-tier and three-tier configurations, involving local and network communication. This figure highlights the server process connection models as opposed to the interaction of the background processes. There is a traditional two-tier client/server connection on the left side, a three-tier connection with an application server on the right side, and a local client connection in the middle of the figure. The two-tier and three-tier connections use a network to communicate with the database, while the local client uses local IPC.

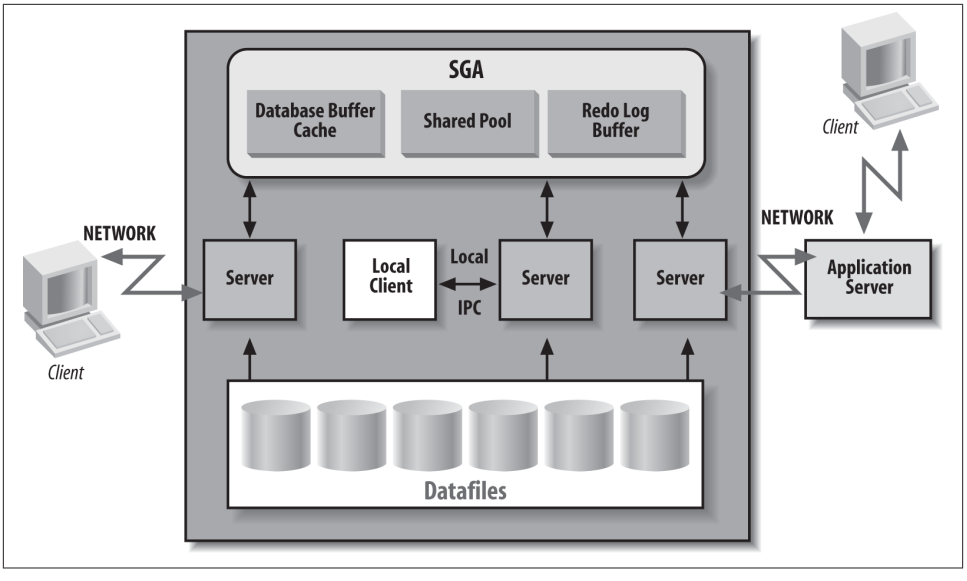


Figure 3-2. Accessing a database

Oracle Net and Establishing Network Connections

The server processes shown in Figure 3-2 are connected to the client processes using some kind of network. How do client processes get connected to Oracle server processes?

The matchmaker that arranges marriages between Oracle clients and server processes is called the Oracle Net Listener. The Listener "listens" for incoming connection requests for one or more instances. The Listener is not part of the Oracle instance—it directs connection requests to the instance. The Listener is started and stopped independently of the instance. If the Listener is down and the instance is up, clients accessing the database over a network cannot find the instance because there is no Listener to guide them. If the Listener is up and the instance is down, there is nowhere to send clients.

The Listener's function is relatively simple:

1. The client contacts the Listener over the network.
2. The Listener detects an incoming request and introduces the requesting client to an Oracle server process.
3. The Listener introduces the server to the client by letting each know the other's network address.
4. The Listener steps out of the way and lets the client and server process communicate directly.

Once the client and the server know how to find each other, they communicate directly. The Listener is no longer required.

Figure 3-3 illustrates the steps outlined above for establishing a networked connection. Network traffic appears as dotted lines.

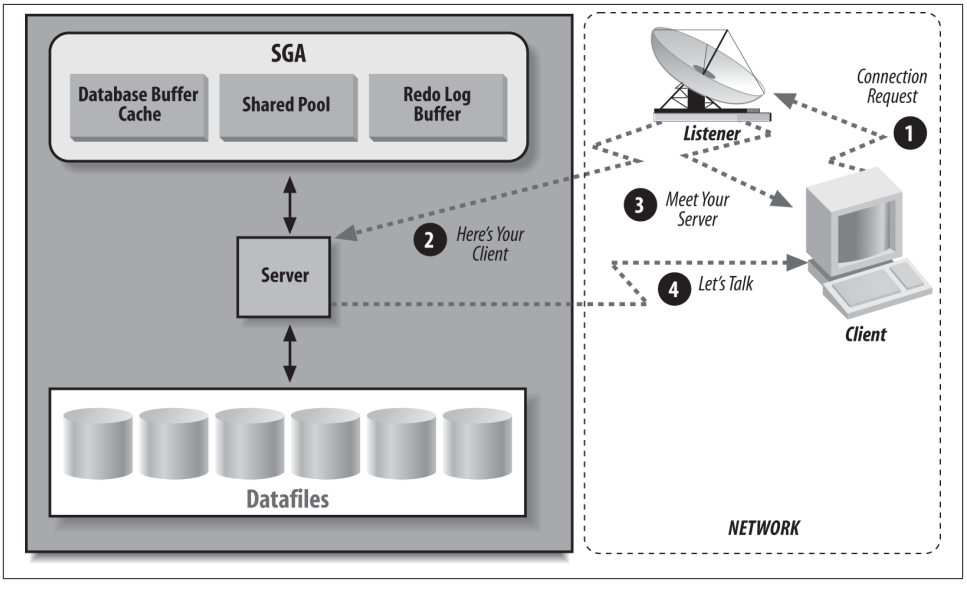


Figure 3-3. Connecting with the Oracle Net Listener

The Shared Server/Multi-Threaded Server

The server processes shown in the diagram are *dedicated*; they serve only one client process. So, if an application has 1,000 clients, the Oracle instance will have 1,000 corresponding server processes. Each server process uses system resources such as the memory and the CPU. Scaling to large user populations can consume a lot of system resources. To support the ever-increasing demand for scalability, Oracle introduced the Multi-Threaded Server (MTS) in Oracle7, known as the *shared server* since Oracle9*i*.

Shared servers allow the Oracle instance to share a set of server processes across a larger group of users. Instead of each client connecting to and using a dedicated server, the clients use shared servers, which can significantly reduce the overall resource requirements for serving large numbers of users.

In many systems there are times when the clients aren't actively using their server process, such as when users are reading and absorbing data retrieved from the database. When a client is not using its server process in the *dedicated model*, that server process still has a hold on system resources even though it isn't doing any useful work. In the

shared server model, the shared server can use the resources of a free client to do work for another client process.

You don't have to make a mutually exclusive choice between shared server processes and dedicated server processes for an Oracle instance. Oracle can mix and match dedicated and shared servers, and clients can connect to one or the other. The choice is based on your Oracle Net configuration files. In the configuration files there will be one service name that leads the client to a dedicated server, and another for connecting via shared servers. The Oracle Net manuals provide the specific syntax for this configuration.

The type of server process a client is using is transparent to the client. From a client perspective, the multithreading or sharing of server processes happens "under the covers," on the database server. The same Listener handles dedicated and multithreaded connection requests.

The steps the Listener takes in establishing a shared server connection are a little different and involve some additional background processes for the instance dispatchers and the shared servers themselves, described here:

Dispatchers
: In the previous description of the Listener, you saw how it forms the connection between a client and server process and then steps out of the way. The client must now be able to depend on a server process that is always available to complete the connection. Because a shared server process may be servicing another client, the client connects to a dispatcher, which is always ready to receive any client request. The dispatchers serve as surrogate dedicated servers for the clients. There are separate dispatchers for each network protocol being used (dispatchers for TCP/IP, etc.). Clients directly connect to their dispatchers instead of to a server process. The dispatchers accept requests from clients and place them in a request queue, which is a memory structure in the SGA. There is one request queue for each instance.

Shared servers
: The shared server processes read from the request queue, process the requests, and place the results in the response queue for the appropriate dispatcher. There is one response queue for each dispatcher. The dispatcher then reads the results from the response queue and sends the information back to the client process.

There is a pool of dispatchers and a pool of shared servers. Oracle starts a certain number of each based on the initialization parameter SHARED_SERVERS that specifies the minimum number of shared servers. Oracle can start additional shared servers up to the value of an optionally specified initialization parameter MAX_SHARED_SERVERS. If Oracle starts additional processes to handle a heavier request load and the load dies down again, Oracle gradually reduces the number of processes to the floor specified by SHARED_SERVERS.

The following steps show how establishing a connection and using shared server processes differ from using a dedicated server process:

1. The client contacts the Listener over the network.
2. The Listener detects an incoming request and, based on the Oracle Net configuration, determines that it is for a multithreaded server. Instead of handing the client off to a dedicated server, the Listener hands the client off to a dispatcher for the network protocol the client is using.
3. The Listener introduces the client and the dispatcher by letting each know the other's network address.
4. Once the client and the dispatcher know where to find each other, they communicate directly. The Listener is no longer required. The client sends each work request directly to the dispatcher.
5. The dispatcher places the client's request in the request queue in the SGA.
6. The next available shared server process reads the request from the request queue and does the work.
7. The shared server places the results for the client's request in the response queue for the dispatcher that originally submitted the request.
8. The dispatcher reads the results from its queue.
9. The dispatcher sends the results to the client.

Figure 3-4 illustrates the steps for using the shared servers. Network traffic appears as dotted lines.

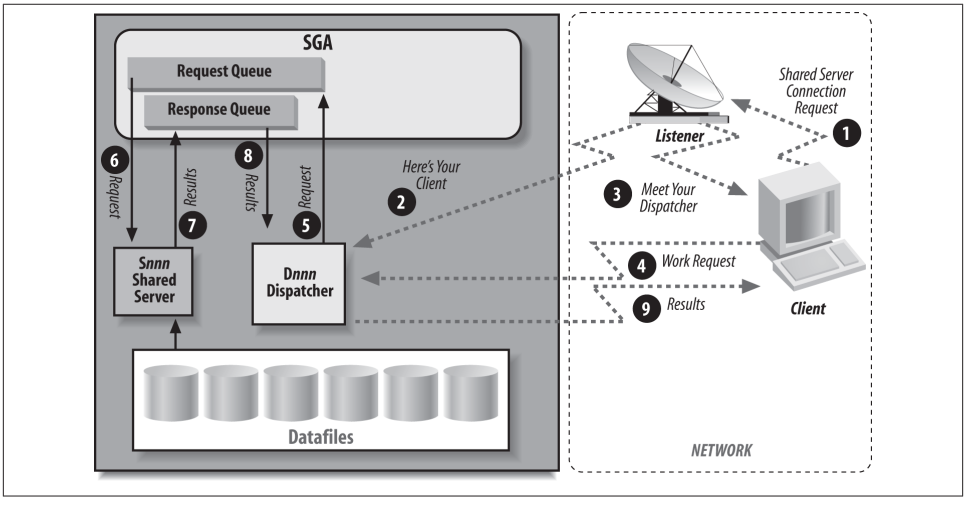


Figure 3-4. Connecting with the Oracle Net Listener (shared servers)

Session memory for shared server processes versus dedicated server processes

There is a concept in Oracle known as *session memory* or *state*. State information is basically data that describes the current status of a session in Oracle. For example, state information contains information about the SQL statements being executed by the session. When you use a dedicated server, this state is stored in the private memory used by the dedicated server. This works out well because the dedicated server works with only one client. The term for this private memory is the Program Global Area (PGA).

If you're using the shared servers, however, any server can work on behalf of a specific client. The session state cannot be stored in the PGA of the shared server process. All servers must be able to access the session state because the session can migrate between different shared servers. For this reason, Oracle places this state information in the System Global Area (SGA).

All servers can read from the SGA. Putting the state information in the SGA allows a session and its state to move from one shared server to another for processing different requests. The server that picks up the request from the request queue simply reads the session state from the SGA, updates the state as needed for processing, and puts it back in the SGA when processing has finished.

The request and response queues, as well as the session state, require additional memory in the SGA, so in older Oracle releases, you would allocate more memory manually if you were using shared servers. By default, the memory for the shared server session state comes from the shared pool. Alternatively, you could also configure something called the *large pool* as a separate area of memory for shared servers. (We introduced the large pool in Chapter 2 in the section "Memory Structures for an Instance" on page 58.) Using the large pool memory avoided the overhead of coordinating memory usage with the shared SQL, dictionary caching, and other functions of the shared pool. This allowed memory management from the large pool and avoided competing with other subsystems for space in and access to the shared pool. Since Oracle Database 10*g*, shared memory is automatically managed by default. Oracle Database 11*g* introduced automated memory management of the SGA and PGA size by default when you set the MEMORY_TARGET initialization parameter, and Oracle Database 12*c* enhanced this automatic memory management of the PGA.

Data dictionary information about the shared server

The data dictionary, which we introduced in Chapter 2, also contains information about the operation of shared servers in the following views:

V$SHARED_SERVER_MONITOR
: This view contains dynamic information about the shared servers, such as high-water marks for connections and how many shared servers have been started and stopped in response to load variations.

V$DISPATCHER
: This view contains details of the dispatcher processes used by the shared server. It can determine how busy the dispatchers are.

V$SHARED_SERVER
: This view contains details of the shared server processes used by the shared server. It can determine how busy the servers are, to help set the floor and ceiling values appropriately.

V$CIRCUIT
: You can think of the route from a client to its dispatcher and from the dispatcher to the shared server (using the queues) as a virtual circuit. This view details these virtual circuits for user connections.

Database Resident Connection Pooling

With Oracle Database 11*g* Release 1, Oracle introduced another type of connection sharing that is even more efficient and scalable than shared servers. As described above, shared servers share a number of server processes, storing session state in the SGA and using a dispatcher to connect requests with an available shared server.

Database resident connection pools are assigned to an application, or can be pooled across multiple applications by connection class. Initially designed for nonthreaded systems, like PHP, a database connection pool keeps a pool of servers available for connection requests, which are assigned by a connection broker (which then gets out of the interaction). When a server is requested, the server remains assigned to the application until the requesting script ends or the connection is explicitly closed.

You can configure multiple pools to handle multiple applications as well as specify the number of servers in each pool and the number of connection brokers.This architecture removes the overhead of having to create and destroy database sessions when connections are opened and closed as well as the creation and destruction of server processes required with dedicated servers. In addition, database resident connection pools eliminate the need for a dispatcher, as with shared servers, and the need to store session state. Due to these differences, database resident connection pooling can provide greater scalability for the highest level of connections to your Oracle Database.

Oracle in the Cloud

This is the fifth edition of this book, the first edition in the era of cloud computing. The Oracle Database is available in several flavors in the cloud, which is the subject of Chapter 15. Much of the discussion of installation and configuration in this chapter is no longer relevant when you are using an Oracle Database in the cloud.

One of the great attractions of using the Oracle Database Cloud, described in detail in Chapter 15, is the ease of getting started. There is no setup or installation of an Oracle Database. You simply request a Database Cloud Service from your browser and, within minutes, you have a working environment available to you with the full power of the Oracle Database.

But just because you don't have to install or configure an Oracle Database Cloud Service does not mean that these tasks were not done. In fact, each Database Cloud Service is implemented with a specific configuration that you cannot change, such as the amount of storage you have selected as well as a variety of configuration options designed to give you adequate computing and I/O without imposing on other users of the Database Cloud.

When you access your Oracle Database Cloud Service, you do not use SQL*Net. You communicate from a browser (or application via RESTful Web Services, described in Chapter 15) with HTTP. The use of this common Internet protocol has two profound implications. First of all, you cannot use SQL*Net over HTTP. This limitation means you cannot simply move data from an on-premise Oracle Database to a Database Cloud Service, modify your *TNSNAMES.ORA* file, and use the same application, as you would if you moved your Oracle Database from one server to another.

Secondly, HTTP is a stateless protocol. This means that you cannot have a transaction span multiple interactions with an Oracle Database Cloud Service. In many cases, this limitation may not make any difference at all, but in some, it may cause you to modify the way you implement your applications.

The Oracle Database Cloud, as well as other ways to use an Oracle Database in the cloud, is the focus of Chapter 15, but for now, understand that although the Oracle Database is certainly well suited to deliver its functionality in a cloud computing environment, the surrounding infrastructure used in the cloud is quite different from a traditional deployment architecture, which means that the great benefits of the cloud will come with some necessary accommodations.

Oracle at Work

To help you truly understand how all the disparate pieces of the Oracle Database work together, this section walks through an example of the steps taken by the Oracle Database to respond to a user request. This example examines the work of a user who is adding new information to the database—in other words, executing a transaction.

Oracle and Transactions

A *transaction* is a work request from a client to retrieve, insert, update, or delete data. (The remainder of this section will focus on transactions that modify data, as opposed to retrieving data.) The statements that change data are a subset of the SQL language

called Data Manipulation Language (DML). Transactions must be handled in a way that guarantees their integrity. Although Chapter 8 delves into transactions more deeply, we must visit a few basic concepts relating to transactions now in order to understand the example in this section:

Transactions are logical and complete
: In database terms, a transaction is a logical unit of work composed of one or more data changes. A transaction may consist of multiple INSERT, UPDATE, and/or DELETE statements affecting data in multiple tables. The entire set of changes must succeed or fail as a complete unit of work. A transaction starts with the first DML statement and ends with either a commit or a rollback.

Oracle also supports autonomous transactions—transactions whose work is committed or rolled back, but that exist within the context of a larger transaction. Autonomous transactions are important because they can commit work without destroying the context of the larger transaction.

Commit or rollback
: Once a user enters the data for his transaction, he can either *commit* the transaction to make the changes permanent or *roll back* the transaction to undo the changes.

System Change Number (SCN)
: A key factor in preserving database integrity is an awareness of which transaction came first. For example, if Oracle is to prevent a later transaction from unwittingly overwriting an earlier transaction's changes, it must know which transaction began first. The mechanism Oracle uses is the System Change Number, a logical timestamp used to track the order in which events occurred. Oracle also uses the SCN to implement multiversion read consistency, which is described in detail in Chapter 8.

Rollback segments
: Rollback segments are structures in the Oracle Database used to store "undo" information for transactions, in case of rollback. This undo information restores database blocks to the state they were in before the transaction in question started. When a transaction starts changing some data in a block, it first writes the old image of the data to a rollback segment. The information stored in a rollback segment is used for two main purposes: to provide the information necessary to roll back a transaction and to support multiversion read consistency.

  A rollback segment is not the same as a redo log. The redo log is used to log all transactions to the database and to recover the database in the event of a system failure, while the rollback segment provides rollback for transactions and read consistency.

Blocks of rollback segments are cached in the SGA just like blocks of tables and indexes. If rollback segment blocks are unused for a period of time, they may be aged out of the cache and written to the disk.

Chapter 8 discusses Oracle's method for concurrency management, multiversion read consistency. This method uses rollback segments to retrieve earlier versions of changed rows. If the required blocks are no longer available, Oracle delivers a "snapshot too old" error.

Oracle9*i* introduced automatic management of rollback segments. In previous versions of the Oracle Database, DBAs had to explicitly create and manage rollback segments. Since Oracle9*i*, you have the ability to specify automatic management of all rollback segments through the use of an undo tablespace. With automatic undo management, you can also specify the length of time that you want to keep undo information; this feature is very helpful if you plan on using flashback queries, discussed in the following section. Oracle Database 10*g* added an undo management retention time advisor.

Fast commits

Because redo logs are written whenever a user commits an Oracle transaction, they can be used to speed up database operations. When a user commits a transaction, Oracle can do one of two things to get the changes into the database on the disk:

- Write all the database blocks the transaction changed to their respective datafiles.
- Write only the redo information, which typically involves much less I/O than writing the database blocks. This recording of the changes can be replayed to reproduce all the transaction's changes later, if they are needed due to a failure.

To provide maximum performance without risking transactional integrity, Oracle writes out only the redo information. When a user commits a transaction, Oracle guarantees that the redo for those changes writes to the redo logs on disk. The actual changed database blocks will be written out to the datafiles later. If a failure occurs before the changed blocks are flushed from the cache to the datafiles, the redo logs will reproduce the changes in their entirety. Because the slowest part of a computer system is the physical disk, Oracle's fast-commit approach minimizes the cost of committing a transaction and provides maximum risk-free performance.

Flashback

In Oracle9*i*, rollback segments were also used to implement a feature called *Flashback Query*. Remember that rollback segments are used to provide a consistent image of the

data in your Oracle Database at a previous point in time. With Flashback Query, you can direct Oracle to return the results for a SQL query at a specific point in time. For instance, you could ask for a set of results from the database as of two hours ago. Flashback provided extra functionality by leveraging the rollback feature that was already a core part of the Oracle architecture.

Since Flashback uses rollback segments, you can only flash back as far as the information in the current rollback segment. This requirement typically limits the span of Flashback to a relatively short period of time—you normally would not be able to roll back days, since your Oracle Database doesn't keep that much rollback information around. Despite this limitation, there are scenarios in which you might be able to use a Flashback Query effectively, such as going back to a point in time before a user made an error that resulted in a loss of data.

The use of Flashback has increased as Oracle has added more flashback capabilities to the database. Oracle Database 10*g* greatly expanded the flashback capabilities available to include:

- Flashback Database, to roll back the entire database to a consistent state
- Flashback Table, to roll back a specific table
- Flashback Drop, to roll back a DROP operation
- Flashback Versions Query, to retrieve changes to one or more rows

Oracle Database 11*g* continued this expansion with the Flashback Transaction feature, which can be used to reverse the effect of a transaction and any other transactions that are dependent on it.Oracle Database 11*g* R2 added Flashback Data Archive, part of the Advanced Compression option.With this feature, older versions of rows are stored in shadow tables, allowing Flashback Queries against very old data without having to have to keep all of the versions in the undo tablespace. In Oracle Database 12*c*, Flashback Queries are extended to support queries on temporal validity dimensions (for valid time periods). Additionally, with Oracle Database 12*c* you can recover an individual table from a database backup.

A Transaction, Step by Step

This simple example illustrates the complete process of a transaction. The example uses the EMP table of employee data, which is part of the traditional test schema shipped with Oracle Databases. In this example, an HR clerk wants to update the name of an employee. The clerk retrieves the employee's data from the database, updates the name, and commits the transaction.

The example assumes that only one user is trying to update the information for a row in the database. Because of this assumption, it won't include the steps normally taken

by Oracle to protect the transaction from changes by other users, which are detailed in Chapter 8.

The HR clerk already has the employee record on-screen and so the database block containing the row for that employee is already in the database buffer cache. The steps from this point would be:

1. The user modifies the employee name on-screen and the client application sends a SQL UPDATE statement over the network to the server process.
2. The server process looks for an identical statement in the shared SQL area of the shared pool. If it finds one, it reuses it. Otherwise, it checks the statement for syntax and evaluates it to determine the best way to execute it. This processing of the SQL statement is called *parsing and optimizing*. (The optimizer is described in more detail in Chapter 4.) Once the processing is done, the statement is cached in the shared SQL area.
3. The server process copies the old image of the employee data about to be changed notes the changes in a rollback segment and to a redo segment. The rollback segment changes are part of the redo. This may seem a bit odd, but remember that redo is generated for all changes resulting from the transaction. The contents of the rollback segment have changed because the old employee data was written to the rollback segment for undo purposes. This change to the contents of the rollback segment is part of the transaction and therefore part of the redo for that transaction.
4. Once the server process has completed this work, the process modifies the database block to change the employee name. The database block is stored in the database cache at this time. Control is passed back to the client process.
5. The HR clerk commits the transaction.
6. The Log Writer (LGWR) process writes the redo information for the entire transaction from the redo log buffer to the current redo logfile on disk. When the operating system confirms that the write to the redo logfile has successfully completed, the transaction is considered committed.
7. The server process sends a message to the client confirming the commit.

The user could have canceled or rolled back the transaction instead of committing it, in which case the server process would have used the old image of the employee data in the rollback segment to undo the change to the database block.

Figure 3-5 shows the steps described here. Network traffic appears as dotted lines.

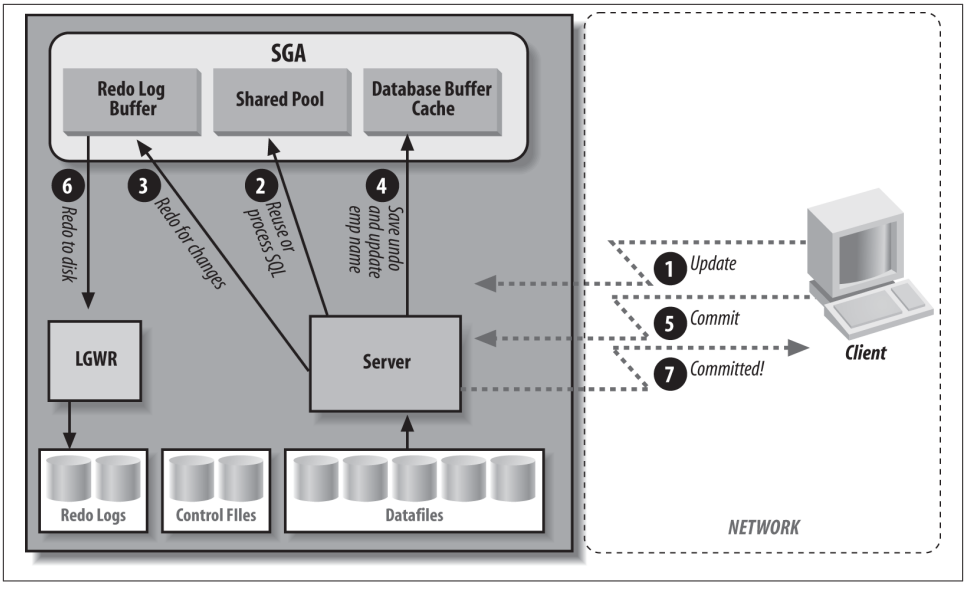


Figure 3-5. Steps for a transaction

CHAPTER 4

Oracle Data Structures

In the previous chapters, we examined some distinctions between the different components that make up an Oracle Database. For example, we pointed out that the Oracle instance differs from the files that make up the physical storage of the data in tablespaces, that you cannot access the data in a tablespace except through an Oracle instance, and that the instance itself isn't very valuable without the data stored in those files.

The instance is the logical entity used by applications and users, separate from the physical storage of data. In a similar way, the actual tables and columns are logical entities within the physical database. The user who makes a request for data from an Oracle Database probably doesn't know anything about instances and tablespaces, but does know about the structure of her data, as implemented with tables and columns. To fully leverage the power of Oracle, you must understand how the Oracle Database server implements and uses these logical data structures, the topic of this chapter.

Datatypes

The *datatype* is one of the attributes for a *column* or a variable in a stored procedure. A datatype describes and limits the type of information stored in a column, and can limit the operations that you can perform on columns.

You can divide Oracle datatype support into three basic varieties: character datatypes, numeric datatypes, and datatypes that represent other kinds of data. You can use any of these datatypes when you create columns in a table, as with this SQL statement:

```
CREATE TABLE SAMPLE_TABLE(
    char_field CHAR(10),
    varchar_field VARCHAR2(10),
    todays_date DATE)
```

You also use these datatypes when you define variables as part of a PL/SQL procedure.

Character Datatypes

Character datatypes can store any string value, including the string representations of numeric values. Assigning a value larger than the length specified or allowed for a character datatype results in a runtime error. You can use string functions, such as UPPER, LOWER, SUBSTR, and SOUNDEX, on standard (not large) character value types.

There are several different character datatypes:

CHAR
: The CHAR datatype stores character values with a fixed length. A CHAR datatype can have between 1 and 2,000 characters. If you don't explicitly specify a length for a CHAR, it assumes the default length of 1. If you assign a value that's shorter than the length specified for the CHAR datatype, Oracle will automatically pad the value with blanks. Some examples of CHAR values are:

```
CHAR(10) = "Rick      ", "Jon       ", "Stackowiak"
```

VARCHAR2
: The VARCHAR2 datatype stores variable-length character strings. Although you must assign a length to a VARCHAR2 datatype, this length is the maximum length for a value rather than the required length. Values assigned to a VARCHAR2 datatype aren't padded with blanks. Because of this, a VARCHAR2 datatype can require less storage space than a CHAR datatype, because the VARCHAR2 datatype stores only the characters assigned to the column. Up until Oracle Database 12*c*, the maximum size of a column with a VARCHAR2 was to 4,000 characters; this limit became 32K in Oracle Database 12*c*.

  At this time, the VARCHAR and VARCHAR2 datatypes are synonymous in Oracle8 and later versions, but Oracle recommends the use of VARCHAR2 because future changes may cause VARCHAR and VARCHAR2 to diverge. The values shown earlier for the CHAR values, if entered as VARCHAR2 values, are:

```
VARCHAR2(10) = "Rick", "Jon", "Stackowiak"
```

NCHAR and NVARCHAR2
: The NCHAR and NVARCHAR2 datatypes store fixed-length or variable-length character data, respectively, using a different character set from the one used by the rest of the database. When you create a database, you specify the character set that will be used for encoding the various characters stored in the database. You can optionally specify a secondary character set as well (which is known as the *National Language Set*, or NLS). The secondary character set will be used for NCHAR and NVARCHAR2 columns. For example, you may have a description field in which you want to store Japanese characters while the rest of the database uses English encoding. You would specify a secondary character set that supports Japanese characters when you create the database, and then use the NCHAR or NVARCHAR2

datatype for the columns in question. The maximum length of NVARCHAR2 columns was also increased to 32K in Oracle Database 12*c*.

Starting with Oracle9*i*, you can specify the length of NCHAR and NVARCHAR2 columns in characters, rather than in bytes. For example, you can indicate that a column with one of these datatypes is 7 characters. The Oracle9*i* database will automatically make the conversion to 14 bytes of storage if the character set requires double-byte storage.

Oracle Database 10*g* introduced the Globalization Development Kit (GDK), which is designed to aid in the creation of Internet applications that will be used with different languages. The key feature of this kit is a framework that implements best practices for globalization for Java and PL/SQL developers.

Oracle Database 10*g* also added support for case- and accent-insensitive queries and sorts. You can use this feature if you want to use only base letters or base letters and accents in a query or sort.

LONG
: The LONG datatype can hold up to 2 GB of character data. It is regarded as a legacy datatype from earlier versions of Oracle. If you want to store large amounts of character data, Oracle now recommends that you use the CLOB and NCLOB datatypes. There are many restrictions on the use of LONG datatypes in a table and within SQL statements, such as the fact that you cannot use LONGs in WHERE, GROUP BY, ORDER BY, or CONNECT BY clauses or in SQL statements with the DISTINCT qualifier. You also cannot create an index on a LONG column.

CLOB and NCLOB
: The CLOB and NCLOB datatypes can store up to 4 GB of character data prior to Oracle Database 10*g*. Starting with Oracle Database 10*g*, the limit has been increased to 128 TBs, depending on the block size of the database. The NCLOB datatype stores the NLS data. Oracle Database 10*g* and later releases implicitly perform conversions between CLOBs and NCLOBs. For more information on CLOBs and NCLOBs, please refer to the discussion about large objects (LOBs) in the section "Other Datatypes" on page 97 later in this chapter.

Numeric Datatype

Oracle uses a standard, variable-length internal format for storing numbers. This internal format can maintain a precision of up to 38 digits.

The numeric datatype for Oracle is NUMBER. Declaring a column or variable as NUMBER will automatically provide a precision of 38 digits. The NUMBER datatype can also accept two qualifiers, as in:

```
column NUMBER( precision, scale )
```

The *precision* of the datatype is the total number of significant digits in the number. You can designate a precision for a number as any number of digits up to 38. If no value is declared for *precision*, Oracle will use a precision of 38. The *scale* represents the number of digits to the right of the decimal point. If no scale is specified, Oracle will use a scale of 0.

If you assign a negative number to the *scale*, Oracle will round the number up to the designated place to the *left* of the decimal point. For example, the following code snippet:

```
column_round NUMBER(10,-2)
column_round = 1,234,567
```

will give `column_round` a value of 1,234,600.

The NUMBER datatype is the only datatype that stores numeric values in Oracle. The ANSI datatypes of DECIMAL, NUMBER, INTEGER, INT, SMALLINT, FLOAT, DOUBLE PRECISION, and REAL are all stored in the NUMBER datatype. The language or product you're using to access Oracle data may support these datatypes, but they're all stored in a NUMBER datatype column.

With Oracle Database 10*g*, Oracle added support for the precision defined in the IEEE 754-1985 standard with the number datatypes of BINARY_FLOAT and BINARY_DOUBLE. Oracle Database 11*g* added support for the number datatype SIMPLE_INTEGER.

Date Datatype

As with the NUMERIC datatype, Oracle stores all dates and times in a standard internal format. The standard Oracle date format for input takes the form of DD-MON-YY HH:MI:SS, where DD represents up to two digits for the day of the month, MON is a three-character abbreviation for the month, YY is a two-digit representation of the year, and HH, MI, and SS are two-digit representations of hours, minutes, and seconds, respectively. If you don't specify any time values, their default values are all zeros in the internal storage.

You can change the format you use for inserting dates for an instance by changing the NLS_DATE_FORMAT parameter for the instance. You can do this for a session by using the ALTER SESSION SQL statement or for a specific value by using parameters with the TO_DATE expression in your SQL statement.

Oracle SQL supports date arithmetic in which integers represent days and fractions represent the fractional component represented by hours, minutes, and seconds. For

example, adding .5 to a date value results in a date and time combination 12 hours later than the initial value. Some examples of date arithmetic are:

```
12-DEC-07 + 10 = 22-DEC-07
31-DEC-2007:23:59:59 + .25 = 1-JAN-2008:5:59:59
```

As of Oracle9*i* Release 2, Oracle also supports two INTERVAL datatypes, INTERVAL YEAR TO MONTH and INTERVAL DAY TO SECOND, which are used for storing a specific amount of time. This data can be used for date arithmetic.

Temporal validity

Oracle Database 12*c* introduced a new concept related to dates called temporal validity. This feature allows you to specify the time period when a particular row of data is valid. For instance, a company might want to add a temporary address for a particular person that would only be valid for a set period of time.

When you create a table, you indicate that the rows of the table will allow for temporal validity, and you can then set a time, as well as a range of time, for each record. Subsequent SQL statements can query for records with a time selection component, which will use the values in the temporal validity columns.

The temporal validity functionality uses Flashback mechanisms for implementation, which are described in Chapter 8.

Other Datatypes

Aside from the basic character, number, and date datatypes, Oracle supports a number of specialized datatypes:

RAW and LONG RAW
: Normally, your Oracle Database not only stores data but also interprets it. When data is requested or exported from the database, the Oracle Database sometimes massages the requested data. For instance, when you dump the values from a NUMBER column, the values written to the dump file are the representations of the numbers, not the internally stored numbers.

  The RAW and LONG RAW datatypes circumvent any interpretation on the part of the Oracle Database. When you specify one of these datatypes, Oracle will store the data as the exact series of bits presented to it. The RAW datatypes typically store objects with their own internal format, such as bitmaps. Until Oracle Database 12*c*, a RAW datatypecolumn could hold 2 KB, which was increased to 32K in that release. Oracle recommends that you convert LONG RAW columns to one of the binary LOB column types.

ROWID
: The ROWID is a special type of column known as a *pseudocolumn*. The ROWID pseudocolumn can be accessed just like a column in a SQL SELECT statement.

There is a ROWID pseudocolumn for every row in an Oracle Database. The ROWID represents the specific address of a particular row. The ROWID pseudocolumn is defined with a ROWID datatype.

The ROWID relates to a specific location on a disk drive. Because of this, the ROWID is the fastest way to retrieve an individual row. However, the ROWID for a row can change as the result of dumping and reloading the database. For this reason, we don't recommend using the value for the ROWID pseudocolumn across transaction lines. For example, there is no reason to store a reference to the ROWID of a row once you've finished using the row in your current application.

You cannot set the value of the standard ROWID pseudocolumn with any SQL statement.

The format of the ROWID pseudocolumn changed with Oracle8. Beginning with Oracle8, the ROWID includes an identifier that points to the database object number in addition to the identifiers that point to the datafile, block, and row. You can parse the value returned from the ROWID pseudocolumn to understand the physical storage of rows in your Oracle Database.

You can define a column or variable with a ROWID datatype, but Oracle doesn't guarantee that any value placed in this column or variable is a valid ROWID.

ORA_ROWSCN
: Oracle Database 10*g* and later releases support a pseudocolumn ORA_ROWSCN, which holds the System Change Number (SCN) of the last transaction that modified the row. You can use this pseudocolumn to check easily for changes in the row since a transaction started. For more information on SCNs, see the discussion of concurrency in Chapter 8.

LOB
: A LOB, or large object datatype, can store up to 4 GB of information. LOBs come in three varieties:

  - CLOB, which can store only character data
  - NCLOB, which stores National Language character set data
  - BLOB, which stores data as binary information

  You can designate that a LOB should store its data within the Oracle Database or that it should point to an external file that contains the data.

  LOBs can participate in transactions. Selecting a LOB datatype from Oracle will return a pointer to the LOB. You must use either the DBMS_LOB PL/SQL built-in package or the OCI interface to actually manipulate the data in a LOB.

To facilitate the conversion of LONG datatypes to LOBs, Oracle9*i* included support for LOBs in most functions that support LONGs, as well as an option to the ALTER TABLE statement that allows the automatic migration of LONG datatypes to LOBs.

BFILE
: The BFILE datatype acts as a pointer to a file stored outside of the Oracle Database. Because of this fact, columns or variables with BFILE datatypes don't participate in transactions, and the data stored in these columns is available only for reading. The file size limitations of the underlying operating system limit the amount of data in a BFILE.

XMLType
: As part of its support for XML, Oracle9*i* introduced a datatype called XMLType. A column defined as this type of data will store an XML document in a character LOB column. There are built-in functions that allow you to extract individual nodes from the document, and you can also build indexes on any particular node in the XMLType document.

Identity datatype
: Oracle Database 12*c* introduces the identity datatype, which matches a datatype found in IBM's DB2 database. This datatype is specifically designed to make it easier to migrate applications that have used DB2 to Oracle.

User-defined data
: Oracle8 and later versions allow users to define their own complex datatypes, which are created as combinations of the basic Oracle datatypes previously discussed. These versions of Oracle also allow users to create objects composed of both basic datatypes and user-defined datatypes. For more information about objects within Oracle, see Chapter 14.

AnyType, AnyData, AnyDataSet
: Oracle9*i* and newer releases include three datatypes that can be used to explicitly define data structures that exist outside the realm of existing datatypes. Each of these datatypes must be defined with program units that let Oracle know how to process any specific implementation of these types.

Type Conversion

Oracle automatically converts some datatypes to other datatypes, depending on the SQL syntax in which the value occurs.

When you assign a character value to a numeric datatype, Oracle performs an implicit conversion of the ASCII value represented by the character string into a number. For instance, assigning a character value such as 10 to a NUMBER column results in an automatic data conversion.

If you attempt to assign an alphabetic value to a numeric datatype, you will end up with an unexpected (and invalid) numeric value, so you should make sure that you're assigning values appropriately.

You can also perform explicit conversions on data, using a variety of conversion functions available with Oracle. Explicit data conversions are better to use if a conversion is anticipated, because they document the conversion and avoid the possibility of going unnoticed, as implicit conversions sometimes do.

Concatenation and Comparisons

The concatenation operator for Oracle SQL on most platforms is two vertical lines (||). Concatenation is performed with two character values. Oracle's automatic type conversion allows you to seemingly concatenate two numeric values. If NUM1 is a numeric column with a value of 1, NUM2 is a numeric column with a value of 2, and NUM3 is a numeric column with a value of 3, the following expressions are TRUE:

```
NUM1 || NUM2 || NUM3 = "123"
NUM1 || NUM2 + NUM3 = "15" (1 || 2 + 3)
NUM1 + NUM2 || NUM3 = "33" (1+ 2 || 3)
```

The result for each of these expressions is a character string, but that character string can be automatically converted back to a numeric column for further calculations.

Comparisons between values of the same datatype work as you would expect. For example, a date that occurs later in time is larger than an earlier date, and 0 or any positive number is larger than any negative number. You can use relational operators to compare numeric values or date values. For character values, comparison of single characters are based on the underlying code pages for the characters. For multicharacter strings, comparisons are made until the first character that differs between the two strings appears.

If two character strings of different lengths are compared, Oracle uses two different types of comparison semantics: *blank-padded comparisons* and *nonpadded comparisons*. For a blank-padded comparison, the shorter string is padded with blanks and the comparison operates as previously described. For a nonpadded comparison, if both strings are identical for the length of the shorter string, the shorter string is identified as smaller. For example, in a blank-padded comparison the string "A " (a capital A followed by a blank) and the string "A" (a capital A by itself) would be seen as equal, because the second value would be padded with a blank. In a nonpadded comparison, the second string would be identified as smaller because it is shorter than the first string. Nonpadded comparisons are used for comparisons in which one or both of the values are VARCHAR2 or NVARCHAR2 datatypes, while blank-padded comparisons are used when neither of the values is one of these datatypes.

Oracle Database 10*g* and later releases include a feature called the Expression Filter, which allows you to store a complex comparison expression as part of a row. You can

use the EVALUATE function to limit queries based on the evaluation of the expression, similar to a normal comparison. The Expression Filter uses regular expressions, which are described later in this chapter.

NULLs

The NULL value is one of the key features of the relational database. The NULL, in fact, doesn't represent any value at all—it represents the lack of a value. When you create a column for a table that must have a value, you specify it as NOT NULL, meaning that it cannot contain a NULL value. If you try to write a row to a database table that doesn't assign a value to a NOT NULL column, Oracle will return an error.

You can assign NULL as a value for any datatype. The NULL value introduces what is called *three-state logic* to your SQL operators. A normal comparison has only two states: TRUE or FALSE. If you're making a comparison that involves a NULL value, there are three logical states: TRUE, FALSE, and neither.

None of the following conditions are true for Column A if the column contains a NULL value:

```
A > 0
A < 0
A = 0
A != 0
```

The existence of three-state logic can be confusing for end users, but your data may frequently require you to allow for NULL values for columns or variables.

You have to test for the presence of a NULL value with the relational operator IS NULL, since a NULL value is not equal to 0 or any other value. Even the expression:

```
NULL = NULL
```

will always evaluate to FALSE, since a NULL value doesn't equal any other value.

Should You Use NULLs?

The idea of three-state logic may seem somewhat confusing, especially when you imagine your poor end users executing ad hoc queries and trying to account for a value that's neither TRUE nor FALSE. This prospect may concern you, so you may decide not to use NULL values at all.

We believe that NULLs have an appropriate use. The NULL value covers a very specific situation: a time when a column has not had a value assigned. The alternative to using a NULL is using a value with another meaning—such as 0 for numbers—and then trying

to somehow determine whether that value has actually been assigned or simply exists as a replacement for NULL.

If you choose not to use NULL values, you're forcing a value to be assigned to a column for every row. You are, in effect, eliminating the possibility of having a column that doesn't require a value, as well as potentially assigning misleading values for certain columns. This situation can be misleading for end users and can lead to inaccurate results for summary actions such as AVG (average).

Avoiding NULL values simply replaces one problem—educating users or providing them with an interface that implicitly understands NULL values—with another set of problems, which can lead to a loss of data integrity.

Basic Data Structures

This section describes the three basic Oracle data structures: tables, views, and indexes. This section discusses partitioning, which affects the way that data in tables and indexes is stored.This section also covers *editions*, a new type of table introduced in Oracle Database 11*g* Release 2.

Tables

The *table* is the basic data structure used in a relational database. A table is a collection of rows. Each *row* in a table contains one or more *columns*. If you're unfamiliar with relational databases, you can map a table to the concept of a file or database in a nonrelational database, just as you can map a row to the concept of a record in a nonrelational database.

As of Oracle9*i*, you can define *external tables*. As the name implies, the data for an external table is stored outside the database, typically in a flat file. The external table is read-only; you cannot update the data it contains. The external table is good for loading and unloading data to files from a database, among other purposes.

Oracle Database 11*g* introduced the ability to create virtual columns for a table. These columns are defined by an expression and, although the results of the expression are not stored, the columns can be accessed by applications at runtime. Oracle Database 12*c* introduces the invisible column, which is stored and maintained like a regular column but is not accessible by user request or considered by the query optimizer.

Editions

Oracle Database 11*g* Release 2 introduced a new concept called an *edition*. An edition is simply a version of a table, and a table can have more than one edition simultaneously. Editions are used to implement *edition-based redefinition* (EBR), useful as table structures evolve over time.

This evolution previously presented a problem when table structures were changed as applications changed. Typically, a new release of an application would be run in test mode, and this testing is most valid if it can run on production data. Prior to EBR, this requirement would force testers to run on a duplicate copy of a potentially large database, and perform their testing outside of a real world production scenario.

With EBR, you can create a new version of a table with a different data structure that resides in the Oracle Database at the same time as a previous version. An application can access a specific edition of a table, so a new version of an application can run simultaneously with a previous one that uses a table with a different structure.

Not only can organizations run two versions of an application at the same time, but developers can roll back the changes in a new version by simply dropping the edition for that new version. In the same way, a new version can be made permanent by dropping a previous version and pointing applications to the newer version, which means an upgrade can take place without any downtime.

You need to do some initialization work to use editions, as well as creating edition-related triggers that will duplicate data manipulation actions in multiple versions.

Views

A *view* is an Oracle data structure defined through a SQL statement. The SQL statement is stored in the database. When you use a view in a query, the stored query is executed and the base table data is returned to the user. Views do not contain data, but represent ways to look at the base table data in the way the query specifies.

You can use a view for several purposes:

- To simplify access to data stored in multiple tables.
- To implement specific security for the data in a table (e.g., by creating a view that includes a WHERE clause that limits the data you can access through the view). Starting with Oracle9*i*, you can use *fine-grained access control* to accomplish the same purpose. Fine-grained access control gives you the ability to automatically limit data access based on the value of data in a row.
- To isolate an application from the specific structure of the underlying tables.

A view is built on a collection of *base tables*, which can be either actual tables in an Oracle Database or other views. If you modify any of the base tables for a view so that they no longer can be used for a view, that view itself can no longer be used.

In general, you can write to the columns of only one underlying base table of a view in a single SQL statement. There are additional restrictions for INSERT, UPDATE, and DELETE operations, and there are certain SQL clauses that prevent you from updating any of the data in a view.

You can write to a nonupdateable view by using an INSTEAD OF trigger, which is described later in this chapter.

Oracle8*i* introduced *materialized views*. These are not really views as defined in this section, but are physical tables that hold pre-summarized data providing significant performance improvements in a data warehouse. Materialized views are described in more detail in Chapter 10.

Indexes

An index is a data structure that speeds up access to particular rows in a database. An index is associated with a particular table and contains the data from one or more columns in the table.

The basic SQL syntax for creating an index is shown in this example:

```
CREATE INDEX emp_idx1 ON emp (ename, job);
```

in which `emp_idx1` is the name of the index, `emp` is the table on which the index is created, and `ename` and `job` are the column values that make up the index.

The Oracle Database server automatically modifies the values in the index when the values in the corresponding columns are modified. Because the index contains less data than the complete row in the table and because indexes are stored in a special structure that makes them faster to read, it takes fewer I/O operations to retrieve the data in them. Selecting rows based on an index value can be faster than selecting rows based on values in the table rows. In addition, most indexes are stored in sorted order (either ascending or descending, depending on the declaration made when you created the index). Because of this storage scheme, selecting rows based on a range of values or returning rows in sorted order is much faster when the range or sort order is contained in the presorted indexes.

In addition to the data for an index, an index entry stores the ROWID for its associated row. The ROWID is the fastest way to retrieve any row in a database, so the subsequent retrieval of a database row is performed in the most optimal way.

An index can be either unique (which means that no two rows in the table or view can have the same index value) or nonunique. If the column or columns on which an index is based contain NULL values, the row isn't included in the index.

An index in Oracle refers to the physical structure used within the database. A *key* is a term for a logical entity, typically the value stored within the index. In most places in the Oracle documentation, the two terms are used interchangeably, with the notable exception of the foreign key constraint, which is discussed later in this chapter.

Four different types of index structures, which are described in the following sections, are used in Oracle: standard B*-tree indexes; reverse key indexes; bitmap indexes; and

function-based indexes, which were introduced in Oracle8*i*. Oracle Database 11*g* delivers the ability to use invisible indexes, which are described below. Oracle also gives you the ability to cluster the data in the tables, which can improve performance. This is described later, in the section "Clusters" on page 112. Finally, something called a *storage index* is available with the Exadata Database Machine, which is not really an index at all, but is described at the end of this section.

B*-tree indexes

The *B*-tree index* is the default index used in Oracle. It gets its name from its resemblance to an inverted tree, as shown in Figure 4-1.

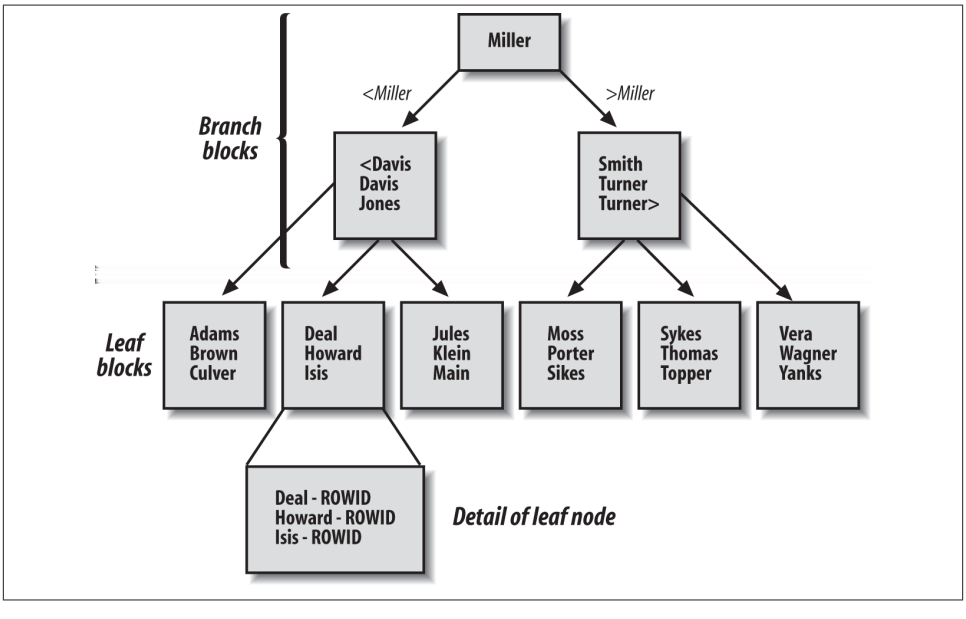


Figure 4-1. A B-tree index*

The B*-tree index is composed of one or more levels of branch blocks and a single level of leaf blocks. The branch blocks contain information about the range of values contained in the next level of branch blocks. The number of branch levels between the root and leaf blocks is called the *depth* of the index. The leaf blocks contain the actual index values and the ROWID for the associated row.

The B*-tree index structure doesn't contain many blocks at the higher levels of branch blocks, so it takes relatively few I/O operations to read quite far down the B*-tree index structure. All leaf blocks are at the same depth in the index, so all retrievals require essentially the same amount of I/O to get to the index entry, which evens out the performance of the index.

Oracle allows you to create *index-organized tables* (IOTs), in which the leaf blocks store the entire row of data rather than only the ROWID that points to the associated row. Index-organized tables reduce the total amount of space needed to store an index and a table by eliminating the need to store the ROWID in the leaf page. But index-organized tables cannot use a UNIQUE constraint or be stored in a cluster. In addition, index organized tables don't support distribution, replication, and partitioning (covered in greater detail in other chapters), although IOTs can be used with Oracle Streams for capturing and applying changes with Oracle Database 10*g* and later releases.

There were a number of enhancements to index-organized tables as of Oracle9*i*, including a lifting of the restriction against the use of bitmap indexes as secondary indexes for an IOT and the ability to create, rebuild, or coalesce secondary indexes on an IOT. Oracle Database 10*g* continued this trend by allowing replication and all types of partitioning for index-organized tables, as well as providing other enhancements.

Reverse key indexes

Reverse key indexes, as their name implies, automatically reverse the order of the bytes in the key value stored in the index. If the value in a row is "ABCD," the value for the reverse key index for that row is "DCBA."

To understand the need for a reverse key index, you have to review some basic facts about the standard B*-tree index. First and foremost, the depth of the B*-tree is determined by the number of entries in the leaf nodes. The greater the depth of the B*-tree, the more levels of branch nodes there are and the more I/O is required to locate and access the appropriate leaf node.

The index illustrated in Figure 4-1 is a nice, well-behaved, alphabetic-based index. It's balanced, with an even distribution of entries across the width of the leaf pages. But some values commonly used for an index are not so well behaved. Incremental values, such as ascending sequence numbers or increasingly later date values, are always added to the right side of the index, which is the home of higher and higher values. In addition, any deletions from the index have a tendency to be skewed toward the left side as older rows are deleted. The net effect of these practices is that over time the index turns into an unbalanced B*-tree, where the left side of the index is more sparsely populated than the leaf nodes on the right side. This unbalanced growth has the overall effect of having an unnecessarily deep B*-tree structure, with the left side of the tree more sparsely populated than the right side, where the new, larger values are stored. The effects described here also apply to the values that are automatically decremented, except that the left side of the B*-tree will end up holding more entries.

You can solve this problem by periodically dropping and re-creating the index. However, you can also solve it by using the reverse value index, which reverses the order of the value of the index. This reversal causes the index entries to be more evenly distributed over the width of the leaf nodes. For example, rather than having the values 234, 235, and 236 be added to the maximum side of the index, they are translated to the values

432, 532, and 632 for storage and then translated back when the values are retrieved. These values are more evenly spread throughout the leaf nodes.

The overall result of the reverse index is to correct the imbalance caused by continually adding increasing values to a standard B*-tree index. For more information about reverse key indexes and where to use them, refer to your Oracle documentation.

Bitmap indexes

In a standard B*-tree index, the ROWIDs are stored in the leaf blocks of the index. In a *bitmap index*, each bit in the index represents a ROWID. If a particular row contains a particular value, the bit for that row is "turned on" in the bitmap for that value. A mapping function converts the bit into its corresponding ROWID. Unlike other index types, bitmap indexes include entries for NULL values.

You can store a bitmap index in much less space than a standard B*-tree index if there aren't many values in the index. Figure 4-2 shows an illustration of how a bitmap index is stored. Figure 10-3 in Chapter 10 shows how a bitmap index is used in a selection condition.

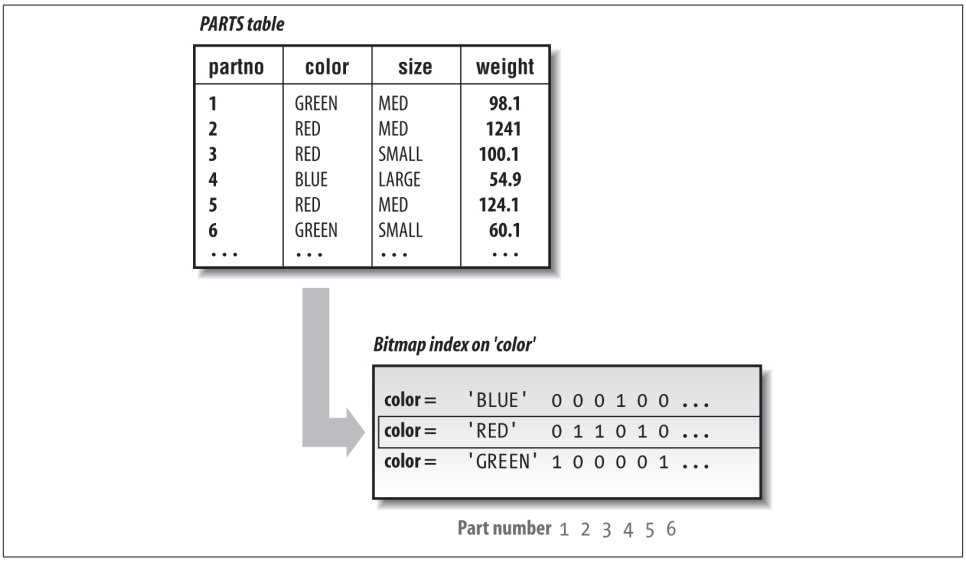


Figure 4-2. Bitmap index

The functionality provided by bitmap indexes is especially important in data warehousing applications in which each dimension of the warehouse contains many repeating values, and queries typically require the interaction of several different dimensions. For more about data warehousing, see Chapter 10.

Function-based indexes

Function-based indexes were introduced in Oracle8*i*. A function-based index is just like a standard B*-tree index, except that you can base the index on the result of a SQL function, rather than just on the value of a column or columns.

Prior to Oracle8*i*, if you wanted to select on the result of a function, Oracle retrieved every row in the database, executed the function, and then accepted or rejected each row. With function-based indexes you can simply use the index for selection, without having to execute the function on every row, every time.

For example, without a function-based index, if you wanted to perform a case-insensitive selection of data you would have to use the UPPER function in the WHERE clause, which would retrieve every candidate row and execute the function. With a function-based index based on the UPPER function, you can select directly from the index.

As of Oracle Database 10*g*, you can perform case- or accent-insensitive queries; these queries provide another way to solve this problem.

This capability becomes even more valuable when you consider that you can create your own functions in an Oracle Database. You can create a very sophisticated function and then create an index based on the function, which can dramatically affect the performance of queries that require the function.

Invisible indexes

Oracle Database 11*g* introduces a new option for all of the index types we've discussed in previous sections—the *invisible index*. Normally, all indexes are used by the optimizer, which is described later in this chapter. You can eliminate an index from optimizer consideration by taking the index offline or by deleting the index. But with both of these methods you will have to take some actions to bring the index up to date when you bring it back into the database environment.

But what if you want to just remove the index from optimizer consideration for a limited time, such as when you are testing performance? With the invisible option, an index is not considered as a possible step in an access path, but updates and deletes to the underlying data are still applied to the index.

Storage indexes

A *storage index* is a structure in Exadata Storage automatically created by the Exadata Storage Server software. A storage index is used to track the highest and lowest values for the most frequently accessed columns in a 1 MB section of storage. These high and low values are stored in memory and used to determine if that 1 MB block should be

accessed as part of a SQL operation.A storage index is created when the 1 MB section is initially accessed.

Storage indexes will have the optimal effect when used on data that is sorted and then loaded. This process would limit the range of values for the sorted columns, which would make the access elimination more efficient. Storage indexes do not negatively impact any SQL, but could have different impacts on a statement based on the storage characteristics of data.

In one sense, a storage index provides the same benefits as a standard index, in that the storage index improves query response time by reducing I/O operations. However, an index is used to navigate to a particular row, while a storage index is used to filter the retrieval of rows needed to satisfy a query.

Partitioning

With the Enterprise Editions of Oracle8 and beyond, you can purchase the Partitioning Option. As the name implies, this option allows you to partition tables and indexes. Partitioning a data structure means that you can divide the information in the structure among multiple physical storage areas. A partitioned data structure is divided based on column values in the table. You can partition tables based on the range of column values in the table (often date ranges), or as the result of a hash function (which returns a value based on a calculation performed on the values in one or more columns). As of Oracle9*i* you can also use a list of values to define a partition, which can be particularly useful in a data warehouse environment.

Oracle Database 11*g* added several partitioning types over its releases. *Interval partitioning* provides the ability to automatically generate a new partition of a fixed interval or range when data to be inserted does not fit into existing partition ranges. *Reference partitioning* is used where a parent-child relationship can be defined between tables and the child table inherits the same partitioning characteristics as the parent. *Virtual column-based partitioning* enables partition keys to be defined by virtual columns.

You can have two levels of partitions, called *composite partitions*, using a combination of partition methods. Prior to Oracle Database 11*g*, you could partition using a composite of range and hash partitioning. Oracle Database 11*g* added the ability to combine list partitioning with list, range, or hash partitioning, or range partitioning with a different range partitioning scheme. Oracle Database 12*c* adds interval reference partitioning.

Oracle is smart enough to take advantage of partitions to improve performance in two ways:

- Oracle won't bother to access partitions that don't contain any data to satisfy the query.

- If all the data in a partition satisfies a part of the WHERE clause for the query, Oracle simply selects all the rows for the partition without bothering to evaluate the clause for each row.

Partitioned tables are especially useful in a data warehouse, in which data can be partitioned based on the time period it spans.

Equally important is the fact that partitioning substantially reduces the scope of maintenance operations and increases the availability of your data. You can perform all maintenance operations, such as backup, recovery, and loading, on a single partition. This flexibility makes it possible to handle extremely large data structures while still performing those maintenance operations in a reasonable amount of time. In addition, if you must recover one partition in a table for some reason, the other partitions in the table can remain online during the recovery operation.

If you have been working with other databases that don't offer the same type of partitioning, you may have tried to implement a similar functionality by dividing a table into several separate tables and then using a `UNION` SQL command to view the data in several tables at once. Partitioned tables give you all the advantages of having several identical tables joined by a `UNION` command without the complexity that implementation requires.

To maximize the benefits of partitioning, it sometimes makes sense to partition a table and an index identically so that both the table partition and the index partition map to the same set of rows. You can automatically implement this type of partitioning, which is called *equipartitioning*, by specifying an index for a partitioned table as a LOCAL index. Local indexes simplify maintenance, since standard operations, such as dropping a partition, will work transparently with both the index partition and the table partition.

Oracle has continued to increase the functionality of partitioning features. Since Oracle Database 10*g* Release 2, you can reorganize individual partitions online, the maximum number of partitions increased from 64 KB – 1 to 128 KB – 1, and query optimization using partition pruning improved.

Oracle Database 11*g* further improved partition pruning, enabled applications to control partitioning, and added a Partition Advisor that can help you to understand when partitioning might improve the performance of your Oracle Database.

Oracle Database 12*c* has added the ability to use partial indexes for a partitioned table. This capability means that you do not have to index all partitions in a table. You can indicate that a particular partition should not have an index, which means that there will not be a local index, or that partition will be excluded from the global index. You can turn indexing on or off for any individual partition.

For more details about the structure and limitations associated with partitioned tables, refer to your Oracle documentation.

Additional Data Structures

There are several other data structures available in your Oracle Database that can be useful in some circumstances.

Sequences

One of the big problems that occurs in a multiuser database is the difficulty of supplying unique numbers for use as keys or identifiers. For this situation, Oracle allows you to create an object called a *sequence*. The sequence object is fairly simple. Whenever anyone requests a value from it, it returns a value and increments its internal value, avoiding contention and time-consuming interaction with the requesting application. Oracle can cache a range of numbers for the sequence so that access to the next number doesn't have to involve disk I/O—the requests can be satisfied from the range in the SGA.

Sequence numbers are defined with a name, an incremental value, and some additional information about the sequence. Sequences exist independently of any particular table, so more than one table can use the same sequence number.

Consider what might happen if you didn't use Oracle sequences. You might store the last sequence number used in a column in a table. A user who wanted to get the next sequence number would read the last number, increment it by a fixed value, and write the new value back to the column. But if many users tried to get a sequence number at the same time, they might all read the "last" sequence number before the new "last" sequence number had been written back. You could lock the row in the table with the column containing the sequence number, but this would cause delays as other users waited on locks. What's the solution? Create a sequence.

Oracle Database 11*g* allowed the use of sequences within PL/SQL expressions. Oracle Database 12*c* added the identity column datatype, which matches the functionality of identity columns in other brands of databases, which used to require conversion to a sequence.

Synonyms

All data structures within an Oracle Database are stored within a specific *schema*. A schema is associated with a particular username, and all objects are referenced with the name of the schema followed by the name of the object.

Schemas are also used as the basis for a form of multitenancy used in Oracle cloud computing, described in Chapter 15.

For instance, if there were a table named EMP in a schema named DEMO, the table would be referenced with the complete name of DEMO.EMP. If you don't supply a specific schema name, Oracle assumes that the structure is in the schema for your current username.

Schemas are a nice feature because object names have to be unique only within their own schemas, but the qualified names for objects can get confusing, especially for end users. To make names simpler and more readable, you can create a *synonym* for any table, view, snapshot, or sequence, or for any PL/SQL procedure, function, or package.

Synonyms can be either *public*, which means that all users of a database can use them, or *private*, which means that only the user whose schema contains the synonym can use it.

For example, if the user DEMO creates a public synonym called EMP for the table EMP in his schema, all other users can simply use EMP to refer to the EMP table in DEMO's schema. Suppose that DEMO didn't create a public synonym and a user called SCOTT wanted to use the name EMP to refer to the EMP table in DEMO's schema. The user SCOTT would create a private synonym in his schema. Of course, SCOTT must have access to DEMO's EMP table for this to work.

Synonyms simplify user access to a data structure. You can also use synonyms to hide the location of a particular data structure, making the data more transportable and increasing the security of the associated table by hiding the name of the schema owner.

Prior to Oracle Database 10*g*, if you changed the location referenced by a synonym, you would have to recompile any PL/SQL procedures that accessed the synonym.

Clusters

A *cluster* is a data structure that improves retrieval performance. A cluster, like an index, does not affect the logical view of the table.

A cluster is a way of storing related data values together on disk. Oracle reads data a block at a time, so storing related values together reduces the number of I/O operations needed to retrieve related values, since a single data block will contain only related rows.

A cluster is composed of one or more tables. The cluster includes a cluster index, which stores all the values for the corresponding cluster key. Each value in the cluster index points to a data block that contains only rows with the same value for the cluster key.

If a cluster contains multiple tables, the tables should be joined together and the cluster index should contain the values that form the basis of the join. Because the value of the cluster key controls the placement of the rows that relate to the key, changing a value in that key can cause Oracle to change the location of rows associated with that key value.

Clusters may not be appropriate for tables that regularly require full table scans, in which a query requires the Oracle Database to iterate through all the rows of the table. Because you access a cluster table through the cluster index, which then points to a data block, full table scans on clustered tables can actually require more I/O operations, lowering overall performance.

Hash Clusters

A *hash cluster* is like a cluster with one significant difference that makes it even faster. Each request for data in a clustered table involves at least two I/O operations, one for the cluster index and one for the data. A hash cluster stores related data rows together, but groups the rows according to a *hash value* for the cluster key. The hash value is calculated with a hash function, which means that each retrieval operation starts with a calculation of the hash value and then goes directly to the data block that contains the relevant rows.

By eliminating the need to go to a cluster index, a hash-clustered table can be even faster for retrieving data than a clustered table. You can control the number of possible hash values for a hash cluster with the HASHKEYS parameter when you create the cluster.

Because the hash cluster directly points to the location of a row in the table, you must allocate all the space required for all the possible values in a hash cluster when you create the cluster.

Hash clusters work best when there is an even distribution of rows among the various values for the hash key. You may have a situation in which there is already a unique value for the hash key column, such as a unique ID. In such situations, you can assign the value for the hash key as the value for the hash function on the unique value, which eliminates the need to execute the hash function as part of the retrieval process. In addition, you can specify your own hash function as part of the definition of a hash cluster.

Oracle Database 10*g* introduced sorted hash clusters, where data is not only stored in a cluster based on a hash value, but is also stored within that location in the order in which it was inserted. This data structure improves performance for applications that access data in the order in which it was added to the database.

Extended Logic for Data

There are several features that have been added to the Oracle Database that are not unique data structures, but rather shape the way you can use the data in the database: the Rules Manager and the Expression Filter.

Rules Manager

The Oracle Database has been continually extending the functionality it can provide, from mere data storage, which still enforced some logical attributes on data, to stored procedures, which allowed you to define extensive logical operations triggered by data manipulations. The Rules Manager, introduced with Oracle Database 10*g* Release 2, takes this extension a step further.

The concept behind the Rules Manager is simple. A *rule* is stored in the database and is called and evaluated by applications. If business conditions or requirements change, the rule covering those scenarios can be changed without having to touch the application code. Rules can be shared across multiple application systems, bringing standardization along with reduced maintenance across the set of applications. You can also create granular rules that can be used in different combinations to implement a variety of conditions.

Rules are invoked by events. The *event* causes the rule to be evaluated and results in a rule action being performed, either immediately or at some later time.

The Rules Manager follows the event-condition action structure and helps users to define five elements required for a Rules Manager application:

- Define an event structure, which is an object in your Oracle Database. Different events have different values for the attributes of the event object.
- Create rules, which include conditions and their subsequent actions.
- Create rule classes to store and group rules with similar structures.
- Create PL/SQL procedures to implement rules.
- Define a results view to configure the rules for external use when the PL/SQL actions cannot be called, such as an application that runs on multiple tiers and has rule actions that are invoked from the application server tier.

You can define conflict resolution routines to handle situations where more than one rule is matched by an event. The Rules Manager also can aggregate different events into composite events and maintain state information until all events are received.

Using Rules can be a very powerful tool for implementing complex logic, but the use of rules can affect your application design. For more information on the Rules Manager, please refer to the Oracle documentation.

The Expression Filter

The Expression Filter, available since Oracle Database 10*g*, uses the Rules Manager to work with expressions. An *expression* is another object type that contains attributes evaluated by the Expression Filter. You add a VARCHAR2 column to a table that stores

the values for the attributes of an expression, use a PL/SQL built-in package to add the expression to the column, and use standard SQL to set the values for the expression. To compare values to an expression, you use the EVALUATE operator in the WHERE clause of your SQL statement.

Expressions can be used to define complex qualities for rows, since an expression can have many attributes. You can also use expressions to implement many-to-many relationships without an intermediary table by using expressions from two tables to join the tables.

With the Enterprise Edition of Oracle, you can add an index to an expression, which can provide the same performance benefits of an index to the values defined as an expression.

Data Design

Tables and columns present a logical view of the data in a relational database. The flexibility of a relational database gives you many options for grouping the individual pieces of data, represented by the columns, into a set of tables. To use Oracle most effectively, you should understand and follow some firmly established principles of database design.

The topic of database design is vast and deep: we won't even pretend to offer more than a cursory overview. However, there are many great books and resources available on the fundamentals of good database design. When E. F. Codd created the concept of a relational database in the 1960s, he also began work on the concept of *normalized* data design. The theory behind normalized data design is pretty straightforward: a table should contain only the information that is directly related to the key value of the table. The process of assembling these logical units of information is called *normalization* of the database design.

Normalized Forms

In fact, there is more than one type of normalization. Each step in the normalization process ends with a specific result called a *normalized form*. There are five standard normalized forms, which are referred to as first normal form (1NF), second normal form (2NF), and so on. The normalization process that we describe briefly in this section results in third normal form (3NF), the most common type of normalization.

Explaining the complete concepts that lie behind the different normal forms is beyond the scope of this chapter and book.

The concept of normalized table design was tailored to the capabilities of the relational database. Because you could join data from different tables together in a query, there

was no need to keep all the information associated with a particular object together in a single record. You could decompose the information into associated units and simply join the appropriate units together when you needed information that crossed table boundaries.

There are many different methodologies for normalizing data. The following is one example:

1. Identify the objects your application needs to know (the *entities*). Examples of entities, as shown in Figure 4-3, include employees, locations, and jobs.
2. Identify the individual pieces of data, referred to by data modelers as *attributes*, for these entities. In Figure 4-3, employee name and salary are attributes. Typically, entities correspond to tables and attributes correspond to columns.
3. As a potential last step in the process, identify *relationships* between the entities based on your business. These relationships are implemented in the database schema through the use of a combination known as a foreign key. For example, the primary key of the DEPARTMENT NUMBER table would be a foreign key column in the EMPLOYEE NAME table used to identify the DEPARTMENT NUMBER for the department in which an employee works. A foreign key is a type of constraint; constraints are discussed later in this chapter.

Normalization provides benefits by avoiding storage of redundant data. Storing the department in every employee record not only would waste space but also would lead to a data maintenance issue. If the department name changed, you would have to update every employee record, even though no employees had actually changed departments. By normalizing the department data into a table and simply pointing to the appropriate row from the employee rows, you avoid both duplication of data and this type of problem.

Normalization also reduces the amount of data that any one row in a table contains. The less data in a row, the less I/O is needed to retrieve it, which helps to avoid this performance bottleneck. In addition, the smaller the size of a row, the more rows are retrieved per data block, which increases the likelihood that more than one desired row will be retrieved in a single I/O operation. And the smaller the row, the more rows will be kept in Oracle's system buffers, which also increases the likelihood that a row will be available in memory when it's needed, thereby avoiding the need for any disk I/O at all.

Finally, the process of normalization includes the creation of foreign key relationships and other data constraints. These relationships build a level of data integrity directly into your database design.

Figure 4-3 shows a simple list of attributes grouped into entities and linked by a foreign key relationship.

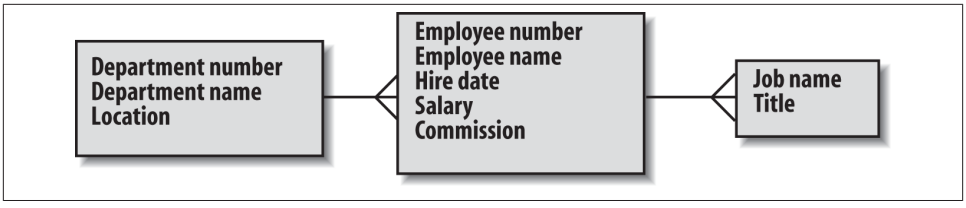


Figure 4-3. The normalization process

However, there is an even more important reason to go through the process of designing a normalized database. You can benefit from normalization because of the planning process that normalizing a data design entails. By really thinking about the way the intended applications use data, you get a much clearer picture of the needs the system is designed to serve. This understanding leads to a much more focused database and application.

Gaining a deep understanding of the way your data will be used also helps with your other design tasks. For instance, once you've completed an optimal logical database design, you can go back and consider what indexes you should add to improve the anticipated performance of the database and whether you should designate any tables as part of a cluster or hash cluster.

Since adding these types of performance-enhancing data structures doesn't affect the logical representation of the database, you can always make these types of modifications later when you see the way an application uses the database in test mode or in production.

Should You Normalize Your Data?

Whenever possible, we recommend that you go through the process of designing a normalized structure for your database.

Data normalization has been proven, both theoretically and in decades of practice, to provide concrete benefits. In addition, the process of creating a normalized data design is intimately intertwined with the process of understanding the data requirements for your application system. You can improve even the simplest database by the discoveries made during the process of normalization.

However, there may be times when you feel that the benefits of a fully normalized design will counteract the performance penalty that a design imposes on your production systems. For example, you may have one, two, or three contact names to be placed in their own table, with a foreign key linking back to the main row for the organization. But because you want to see all the contact names every time you request contact information, you might decide to save the overhead and added development effort of the join

and simply include the three contact names in your organization table. This technique is common in decision-support/data warehousing applications.

Of course, this violation of the rules of normalization limits the flexibility of your application systems—for example, if you later decide that you need four contact names, some modification of every application and report that uses the contact names will be necessary. Normalization leads to a more flexible design, which is a good thing in the constantly changing world we live in.

For this reason, we suggest that you always implement a fully normalized database design and then, if necessary, go back and denormalize certain tables as needed. With this approach, you will at least have to make a conscious decision to "break" the normalization, which involves an active consideration of the price of denormalization.

Constraints

A *constraint* enforces certain aspects of data integrity within a database. When you add a constraint to a particular column, Oracle automatically ensures that data violating that constraint is never accepted. If a user attempts to write data that violates a constraint, Oracle returns an error for the offending SQL statement. Because a constraint is directly associated with the data it is constraining, the restriction is always enforced, eliminating the need to implement this integrity restriction in one or more other locations, such as multiple applications or access tools.

Constraints may be associated with columns when you create or add the table containing the column (via a number of keywords) or after the table has been created with the SQL command ALTER TABLE. Since Oracle8, the following constraint types are supported:

NOT NULL
: You can designate any column as NOT NULL. If any SQL operation leaves a NULL value in a column with a NOT NULL constraint, Oracle returns an error for the statement.

Unique
: When you designate a column or set of columns as unique, users cannot add values that already exist in another row in the table for those columns, or modify existing values to match other values in the column.

  The unique constraint is implemented by a creation of an index, which requires a unique value. If you include more than one column as part of a unique key, you will create a single index that will include all the columns in the unique key. If an index already exists for this purpose, Oracle will automatically use that index.

  If a column is unique but allows NULL values, any number of rows can have a NULL value, because the NULL indicates the absence of a value. To require a truly unique value for a column in every row, the column should be both unique and NOT NULL.

Primary key
: Each table can have, at most, a single primary key constraint. The primary key may consist of more than one column in a table.

  The primary key constraint forces each primary key to have a unique value. It enforces both the unique constraint and the NOT NULL constraint. A primary key constraint will create a unique index, if one doesn't already exist for the specified column(s).

Foreign key
: The foreign key constraint is defined for a table (known as the *child*) that has a relationship with another table in the database (known as the *parent*). The value entered in a foreign key must be present in a unique or primary key of the parent table. For example, the column for a department ID in an employee table might be a foreign key for the department ID primary key in the department table.

  A foreign key can have one or more columns, but the referenced key must have an equal number of columns. You can have a foreign key relate to the primary key of its own table, such as when the employee ID of a manager is a foreign key referencing the ID column in the same table.

  A foreign key can contain a NULL value if it's not forbidden through another constraint.

  By requiring that the value for a foreign key exist in another table, the foreign key constraint enforces referential integrity in the database. Foreign keys not only provide a way to join related tables but also ensure that the relationship between the two tables will have the required data integrity.

  Normally, you would not allow applications to delete or update a row in a parent table if it causes a row in the child table to violate a foreign key constraint. However, you can specify that a foreign key constraint causes a *cascade delete*, which means that deleting a referenced row in the parent table automatically deletes all rows in the child table that reference the primary key value in the deleted row in the parent table. In addition, you could define the constraint to set the value of a corresponding child key to NULL if the parent key value is deleted.

Check
: A check constraint is a more general purpose constraint. A check constraint is a Boolean expression that evaluates to either TRUE or FALSE. If the check constraint evaluates to FALSE, the SQL statement that caused the result returns an error. For example, a check constraint might require the minimum balance in a bank account to be over $100. If a user tries to update data for that account in a way that causes the balance to drop below this required amount, the constraint will return an error.

Some constraints require the creation of indexes to support them. For instance, the unique constraint creates an implicit index used to guarantee uniqueness. You can also specify a particular index that will enforce a constraint when you define that constraint.

All constraints can be either immediate or deferred. An *immediate constraint* is enforced as soon as a write operation affects a constrained column in the table. A *deferred constraint* is enforced when the SQL statement that caused the change in the constrained column completes. Because a single SQL statement can affect several rows, the choice between using a deferred constraint or an immediate constraint can significantly affect how the integrity dictated by the constraint operates. You can specify that an individual constraint is immediate or deferred, or you can set the timing for all constraints in a single transaction.

Finally, you can temporarily suspend the enforcement of constraints for a particular table. When you enable the operation of the constraint, you can instruct Oracle to validate all the data for the constraint or simply start applying the constraint to the new data. When you add a constraint to an existing table, you can also specify whether you want to check all the existing rows in the table.

Triggers

You use constraints to automatically enforce data integrity rules whenever a user tries to write or modify a row in a table. There are times when you want to use the same kind of timing for your own application-specific logic. Oracle includes *triggers* to give you this capability.

Although you can write triggers to perform the work of a constraint, Oracle has optimized the operation of constraints, so it's best to always use a constraint instead of a trigger if possible.

A trigger is a block of code that is fired whenever a particular type of database event occurs to a table. There are three types of common events that can cause a trigger to fire:

- A database UPDATE
- A database INSERT
- A database DELETE

You can, for instance, define a trigger to write a customized audit record whenever a user changes a row.

Triggers are defined at the row level. You can specify that a trigger be fired for each row or for the SQL statement that fires the trigger event. As with the previous discussion of constraints, a single SQL statement can affect many rows, so the specification of the trigger can have a significant effect on the operation of the trigger and the performance of the database.

There are three times when a trigger can fire:

- Before the execution of the triggering event
- After the execution of the triggering event
- Instead of the triggering event

Combining the first two timing options with the row and statement versions of a trigger gives you four possible trigger implementations: before a statement, before a row, after a statement, and after a row.

Oracle Database 11*g* introduced the concept of compound triggers; with this enhancement, a single trigger can have a section for different timing implementations. Compound triggers help to improve performance, since the trigger has to be loaded only once for multiple timing options.

INSTEAD OF triggers were introduced with Oracle8. The INSTEAD OF trigger has a specific purpose: to implement data manipulation operations on views that don't normally permit them, such as a view that references columns in more than one base table for updates. You should be careful when using INSTEAD OF triggers because of the many potential problems associated with modifying the data in the underlying base tables of a view. There are many restrictions on when you can use INSTEAD OF triggers. Refer to your Oracle documentation for a detailed description of the forbidden scenarios.

You can specify a *trigger restriction* for any trigger. A trigger restriction is a Boolean expression that circumvents the execution of the trigger if it evaluates to FALSE.

Triggers are defined and stored separately from the tables that use them. Since they contain logic, they must be written in a language with capabilities beyond those of SQL, which is designed to access data. Oracle8 and later versions allow you to write triggers in PL/SQL, the procedural language that has been a part of Oracle since Version 6. Oracle8*i* and beyond also support Java as a procedural language, so you can create Java triggers with those versions.

You can write a trigger directly in PL/SQL or Java, or a trigger can call an existing stored procedure written in either language.

Triggers are fired as a result of a SQL statement that affects a row in a particular table. It's possible for the actions of the trigger to modify the data in the table or to cause changes in other tables that fire their own triggers. The end result of this may be data

that ends up being changed in a way that Oracle thinks is logically illegal. These situations can cause Oracle to return runtime errors referring to *mutating tables*, which are tables modified by other triggers, or *constraining tables*, which are tables modified by other constraints. Oracle8*i* eliminated some of the errors caused by activating constraints with triggers.

Oracle8*i* also introduced a very useful set of system event triggers (sometimes called *database-level event triggers*), and user event triggers (sometimes called *schema-level event triggers*). For example, you can place a trigger on system events such as database startup and shutdown and on user events such as logging on and logging off.

Query Optimization

All of the data structures discussed so far in this chapter are database entities. Users request data from an Oracle server through database queries. Oracle's query optimizer must then determine the best way to access the data requested by each query.

One of the great virtues of a relational database is its ability to access data without predefining the access paths to the data. When a SQL query is submitted to an Oracle Database, Oracle must decide how to access the data. The process of making this decision is called *query optimization*, because Oracle looks for the optimal way to retrieve the data. This retrieval is known as the *execution path*. The trick behind query optimization is to choose the most efficient way to get the data, since there may be many different options available.

For instance, even with a query that involves only a single table, Oracle can take either of these approaches:

- Use an index to find the ROWIDs of the requested rows and then retrieve those rows from the table.
- Scan the table to find and retrieve the rows; this is referred to as a *full table scan*.

Although it's usually much faster to retrieve data using an index, the process of getting the values from the index involves an additional I/O step in processing the query. This additional step could mean that there were more total I/Os involved to satisfy the query —if, for instance, all the rows in a table were being selected. Query optimization may be as simple as determining whether the query involves selection conditions that can be imposed on values in the index. Using the index values to select the desired rows involves less I/O and is therefore more efficient than retrieving all the data from the table and then imposing the selection conditions. But when you start to consider something as simple as what percent of the rows in a table will be eliminated by using an index, you can see that the complexity in selecting the right execution path can grow very complex very rapidly in a production scenario.

Another factor in determining the optimal query execution plan is whether there is an ORDER BY condition in the query that can be automatically implemented by the presorted index. Alternatively, if the table is small enough, the optimizer may decide to simply read all the blocks of the table and bypass the index since it estimates the cost of the index I/O plus the table I/O to be higher than just the table I/O.

The query optimizer has to make some key decisions even with a query on a single table. When a more involved query is submitted, such as one involving many tables that must be joined together efficiently, or one that has complex selection criteria and multiple levels of sorting, the query optimizer has a much more complex task.

Prior to Oracle Database 10*g*, you could choose between two different Oracle query optimizers, a *rule-based optimizer* and a *cost-based optimizer*; these are described in the following sections. Since Oracle Database 10*g*, the rule-based optimizer is desupported. The references to syntax and operations for the rule-based optimizer in the following sections are provided for reference and are applicable only if you are running a very old release of Oracle.

Rule-Based Optimization

Oracle has always had a query optimizer, but until Oracle7 the optimizer was only rule based. The rule-based optimizer, as the name implies, uses a set of predefined rules as the main determinant of query optimization decisions. Since the rule-based optimizer has been desupported as of Oracle Database 10*g*, your interest in this topic is likely be limited to supporting old Oracle Databases where this choice may have been made.

Rule-based optimization sometimes provided better performance than the early versions of Oracle's cost-based optimizer for specific situations. The rule-based optimizer had several weaknesses, including offering only a simplistic set of rules—and, at the time of this writing, has not been enhanced for several releases. The Oracle rule-based optimizer had about 20 rules and assigned a weight to each one of them. In a complex database, a query can easily involve several tables, each with several indexes and complex selection conditions and ordering. This complexity means that there were a lot of options, and the simple set of rules used by the rule-based optimizer might not differentiate the choices well enough to make the best choice.

The rule-based optimizer assigned an optimization score to each potential execution path and then took the path with the best optimization score. Another weakness in the rule-based optimizer was resolution of optimization choices made in the event of a "tie" score. When two paths presented the same optimization score, the rule-based optimizer looked to the syntax of the SQL statement to resolve the tie. The winning execution path was based on the order in which the tables occured in the SQL statement.

You can understand the potential impact of this type of tiebreaker by looking at a simple situation in which a small table with 10 rows, SMALLTAB, is joined to a large table with

10,000 rows, LARGETAB, as shown in Figure 4-4. If the optimizer chose to read SMALLTAB first, the Oracle Database would read the 10 rows and then read LARGETAB to find the matching rows for each of the 10 rows. If the optimizer chose to read LARGETAB first, the database would read 10,000 rows from LARGETAB and then read SMALLTAB 10,000 times to find the matching rows. Of course, the rows in SMALLTAB would probably be cached, reducing the impact of each probe, but you could still see a dramatic difference in performance.

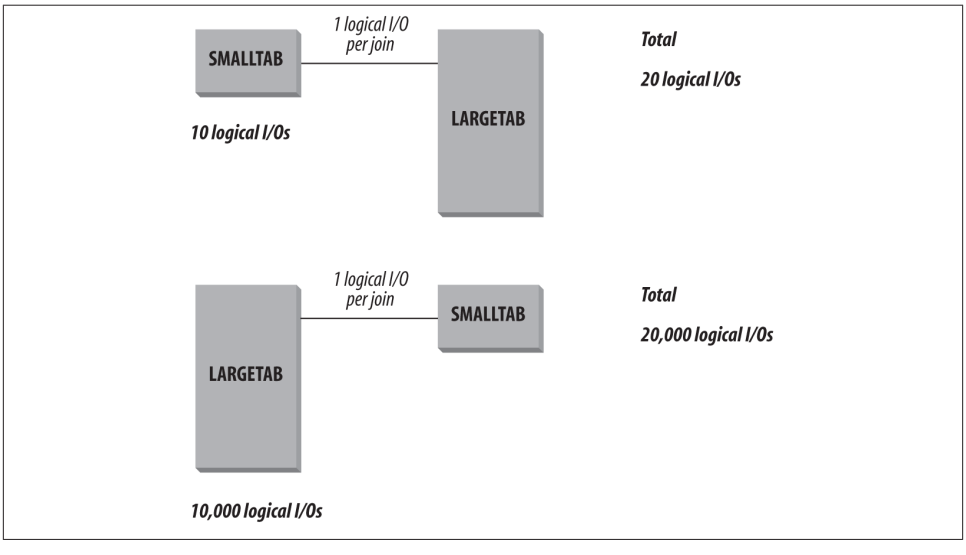


Figure 4-4. The effect of optimization choices

Differences like this could occur with the rule-based optimizer as a result of the ordering of the table names in the query. In the previous situation the rule-based optimizer returned the same results for the query, but it used widely varying amounts of resources to retrieve those results.

Cost-Based Optimization

To improve the optimization of SQL statements, Oracle introduced the *cost-based optimizer* in Oracle7. As the name implies, the cost-based optimizer does more than simply look at a set of optimization rules; instead, it selects the execution path that requires the least number of logical I/O operations. This approach avoids the problems discussed in the previous section. The cost-based optimizer would know which table was bigger and would select the right table to begin the query, regardless of the syntax of the SQL statement.

Oracle8 and later versions, by default, use the cost-based optimizer to identify the optimal execution plan. And, since Oracle Database 10*g*, the cost-based optimizer is the only supported optimizer. To properly evaluate the cost of any particular execution plan, the cost-based optimizer uses statistics about the composition of the relevant data structures. These statistics are automatically gathered by default since the Oracle Database 10*g* release. Among the statistics gathered in the AWR are database segment access and usage statistics, time model statistics, system and session statistics, statistics about which SQL statements that produce the greatest loads, and Active Session History (ASH) statistics.

How statistics are used

The cost-based optimizer finds the optimal execution plan by assigning an optimization score for each of the potential execution plans using its own internal rules and logic along with statistics that reflect the state of the data structures in the database. These statistics relate to the tables, columns, and indexes involved in the execution plan. The statistics for each type of data structure are listed in Table 4-1.

Table 4-1. Database statistics

Data structure	Type of statistics
Table	Number of rows Number of blocks Number of unused blocks Average available free space per block Number of chained rows Average row length
Column	Number of distinct values per column Second-lowest column value Second-highest column value Column density factor
Index	Depth of index B*-tree structure Number of leaf blocks Number of distinct values Average number of leaf blocks per key Average number of data blocks per key Clustering factor

Oracle Database 10*g* and more current database releases also collect overall system statistics, including I/O and CPU performance and utilization. These statistics are stored in the data dictionary, described in this chapter's final section, "Data Dictionary Tables" on page 134.

You can see that these statistics can be used individually and in combination to determine the overall cost of the I/O required by an execution plan. The statistics reflect both the size of a table and the amount of unused space within the blocks; this space can, in turn, affect how many I/O operations are needed to retrieve rows. The index statistics

reflect not only the depth and breadth of the index tree, but also the uniqueness of the values in the tree, which can affect the ease with which values can be selected using the index.

The accuracy of the cost-based optimizer depends on the accuracy of the statistics it uses, so updating statistics has always been a must. Formerly, you would have used the SQL statement ANALYZE to compute or estimate these statistics. When managing an older release, many database administrators also used a built-in PL/SQL package, DBMS_STATS, which contains a number of procedures that helped automate the process of collecting statistics.

Stale statistics can lead to database performance problems, which is why database statistics gathering has been automated by Oracle. This statistics gathering can be quite granular. For example, as of Oracle Database 10*g*, you can enable automatic statistics collection for a table, which can be based on whether a table is either stale (which means that more than 10 percent of the objects in the table have changed) or empty.

The optimizer in Oracle Database 12*c* now automatically decides whether available statistics can generate a good execution plan during the compilation of SQL statements. If statistics are missing or out of date, dynamic sampling of tables automatically occurs to generate new statistics. Also in this database release, statistics are automatically created as part of bulk loading operations.

The use of statistics makes it possible for the cost-based optimizer to make a much more well-informed choice of the optimal execution plan. For instance, the optimizer could be trying to decide between two indexes to use in an execution plan that involves a selection based on a value in either index. The rule-based optimizer might very well rate both indexes equally and resort to the order in which they appear in the WHERE clause to choose an execution plan. The cost-based optimizer, however, knows that one index contains 1,000 entries while the other contains 10,000 entries. It even knows that the index that contains 1,000 values contains only 20 unique values, while the index that contains 10,000 values has 5,000 unique values. The selectivity offered by the larger index is much greater, so that index will be assigned a better optimization score and used for the query.

Testing the Effect of New Statistics

There may be times when you don't want to update your statistics, such as when the distribution of data in your database has reached a steady state or when your queries are already performing optimally (or at least deliver adequate, consistent performance).

Oracle gives you a way to try out a new set of statistics to see if they might make things better while still maintaining the option of returning to the old set: you can save your statistics in a separate table and then collect new ones. If, after testing your application with these new statistics, you decide you preferred the way the old statistics worked, you can simply reload the saved statistics.

In Oracle9*i*, you have the option of allowing the cost-based optimizer to use CPU speed as one of the factors in determining the optimal execution plan. An initialization parameter turns this feature on and off. As of Oracle Database 10*g*, the default cost basis is calculated on the CPU cost plus the I/O cost for a plan.

Even with all the information available to it, the cost-based optimizer did have some noticeable initial flaws. Aside from the fact that it (like all software) occasionally had bugs, the cost-based optimizer used statistics that didn't provide a complete picture of the data structures. In the previous example, the only thing the statistics tell the optimizer about the indexes is the number of distinct values in each index. They don't reveal anything about the distribution of those values. For instance, the larger index can contain 5,000 unique values, but these values can each represent two rows in the associated table, or one index value can represent 5,001 rows while the rest of the index values represent a single row. The selectivity of the index can vary wildly, depending on the value used in the selection criteria of the SQL statement. Fortunately, Oracle 7.3 introduced support for collecting histogram statistics for indexes to address this exact problem. You could create histograms using syntax within the `ANALYZE INDEX` command when you gathered statistics yourself in Oracle versions prior to Oracle Database 10*g*. This syntax is described in your Oracle SQL reference documentation.

Influencing the cost-based optimizer

There are two ways you can influence the way the cost-based optimizer selects an execution plan. The first way is by setting the OPTIMIZER_MODE initialization parameter. ALL_ROWS is the default setting for OPTIMIZER_MODE, enabling optimization with the goal of best throughput. FIRST_ROWS optimizes plans for returning the first set of rows from a SQL statement. You can specify the number of rows using this parameter. The optimizer mode tilts the evaluation of optimization scores slightly and, in some cases, may result in a different execution plan.

Oracle also gives you a way to influence the decisions of the optimizer with a technique called *hints*. A hint is nothing more than a comment with a specific format inside a SQL statement. Hints can be categorized as follows:

- Optimizer SQL hints for changing the query optimizer goal
- Full table scan hints
- Index unique scan hints

- Index range scan descending hints
- Fast full index scan hints
- Join hints, including index joins, nested loop joins, hash joins, sort merge joins, Cartesian joins, and join order
- Other optimizer hints, including access paths, query transformations, and parallel execution

Hints come with their own set of problems. A hint looks just like a comment, as shown in this extremely simple SQL statement. Here, the hint forces the optimizer to use the EMP_IDX index for the EMP table:

```
SELECT /*+ INDEX(EMP_IDX) */ LASTNAME, FIRSTNAME, PHONE FROM EMP
```

If a hint isn't in the right place in the SQL statement, if the hint keyword is misspelled, or if you change the name of a data structure so that the hint no longer refers to an existing structure, the hint will simply be ignored, just as a comment would be. Because hints are embedded into SQL statements, repairing them can be quite frustrating and time-consuming if they aren't working properly. In addition, if you add a hint to a SQL statement to address a problem caused by a bug in the cost-based optimizer and the cost-based optimizer is subsequently fixed, the SQL statement will still not use the corrected (and potentially improved) optimizer.

However, hints do have a place—for example, when a developer has a user-defined datatype that suggests a particular type of access. The optimizer cannot anticipate the effect of user-defined datatypes, but a hint can properly enable the appropriate retrieval path.

For more details about when hints might be considered, see the sidebar "Accepting the Verdict of the Optimizer" on page 129 later in this chapter.

Specifying an Optimizer Mode

In the previous section, we mentioned two optimizer modes: ALL_ROWS and FIRST_ROWS. Two other optimizer modes for Oracle versions prior to Oracle Database 10*g* were:

RULE
: Forces the use of the rule-based optimizer

CHOOSE
: Allowed Oracle to choose whether to use the cost-based optimizer or the rule-based optimizer

With an optimizer mode of CHOOSE, which previously was the default setting, Oracle would use the cost-based optimizer if any of the tables in the SQL statement had statistics associated with them. The cost-based optimizer would make a statistical estimate for

the tables that lacked statistics. In the years since the cost-based optimizer was introduced, the value of using costs has proven itself as this version of the optimizer has grown in accuracy and sophistication, as the next section and the remainder of this chapter illustrate.

Newer database releases and the cost-based optimizer

The cost-based optimizer makes decisions with a wider range of knowledge about the data structures in the database than the previous rule-based optimizer. Although the cost-based optimizer isn't flawless in its decision-making process, it does make more accurate decisions based on its wider base of information, especially because it has matured since its introduction in Oracle7 and has improved with each new release.

The cost-based optimizer also takes into account improvements and new features in the Oracle Database as they are released. For instance, the cost-based optimizer understands the impact that partitioned tables have on the selection of an execution plan, while the rule-based optimizer does not. The cost-based optimizer optimizes execution plans for star schema queries, heavily used in data warehousing, while the rule-based optimizer has not been enhanced to deal effectively with these types of queries or leverage many other such business intelligence query features.

Oracle Corporation was quite frank about its intention to make the cost-based optimizer *the* optimizer for the Oracle Database through a period of years when both optimizer types were supported. In fact, since Oracle Database 10*g*, the rule-based optimizer is no longer supported.

We will remind you of one fact of database design at this point. As good as the cost-based optimizer is today, it is not a magic potion that remedies problems brought on by a poor database and application design or a badly selected hardware and storage platform. When performance problems occur today, they are most often due to bad design and deployment choices.

Accepting the Verdict of the Optimizer

Some of you may doubt the effectiveness of Oracle query optimization if you are on an old Oracle Database release prior to Oracle Database 10*g*. You may have seen cases in which the query optimizer chose an incorrect execution path that resulted in poor performance. You may feel that you have a better understanding of the structure and use of the database than the query optimizer. For these reasons, you might look to hints to force the acceptance of the execution path you feel is correct.

We recommend using the query optimizer for all of your queries rather than using hints. Although the Oracle developers who wrote the query optimizer had no knowledge of your particular database, they did depend on a lot of customer feedback, experience, and knowledge of how Oracle processes queries during the creation of the query optimizer. They designed the cost-based optimizer to efficiently execute all types of queries

that may be submitted to the Oracle Database. And, oh yes, they have been in a process of continual improvement for over 15 years.

In addition, there are three advantages that the query optimizer has over your discretion in all cases:

- The optimizer sees the structure of the entire database. Many Oracle Databases support a variety of applications and users and it's quite possible that your system shares data with other systems, making the overall structure and composition of the data somewhat out of your control. In addition, you probably designed and tested your systems in a limited environment, so your idea of the optimal execution path may not match the reality of the production environment, especially as it evolves.
- The optimizer has a dynamically changing view of the database and its data. The statistics used by the cost-based optimizer can change with each automated collection. In addition to the changing statistical conditions, the internal workings of the optimizer are occasionally changed to accommodate changes in the way the Oracle Database operates. Since Oracle9*i*, the cost-based optimizer takes into account the speed of the CPU, and since Oracle Database 10*g*, leverages statistics on I/O. If you force the selection of a particular query plan with a hint, you might not benefit from changes in Oracle.
- A bad choice by the optimizer may be a sign that something is amiss in your database. In the overwhelming majority of cases, the optimizer selects the optimal execution path. What may be seen as a mistake by the query optimizer can, in reality, be traced to a misconception about the database and its design or to an improper implementation. A mistake is always an opportunity to learn, and you should always take advantage of any opportunity to increase your overall understanding of how Oracle and its optimizer work.

We recommend that you consider using hints only when you have determined them to be absolutely necessary by thoroughly investigating the causes for an optimization problem. The hint syntax was included in Oracle syntax as a way to handle exceptional situations, rather than to allow you to circumvent the query optimizer. If you've found a performance anomaly and further investigation has led to the discovery that the query optimizer is choosing an incorrect execution path, then and only then should you assign a hint to a query. In other words, do not use a hint until you can explain why the optimizer made a poor choice in the first place.

Even in this situation, we recommend that you keep an eye on the hinted query in a production environment to make sure that the forced execution path is still working optimally.

Saving the Optimization

There may be times when you want to prevent the optimizer from calculating a new plan whenever a SQL statement is submitted. For example, you might do this if you've finally reached a point at which you feel the SQL is running optimally, and you don't want the plan to change regardless of future changes to the optimizer or the database.

Starting with Oracle8*i*, you could create a *stored outline* that stored the attributes used by the optimizer to create an execution plan. Once you had a stored outline, the optimizer simply used the stored attributes to create an execution plan. As of Oracle9*i*, you could also edit the hints that were in the stored outline.

With the release of Oracle Database 11*g*, Oracle suggested that you move your stored outlines to *SQL plan baselines*. Now, in addition to manually loading plans, Oracle can be set to automatically capture plan histories into these SQL plan baselines. Included in this gathered history is the SQL text, outline, bind variables, and compilation environment. When a SQL statement is compiled, Oracle will first use the cost-based optimizer to generate a plan and will evaluate any matching SQL plan baselines for relative cost, choosing the plan with the lowest cost.

Oracle Database 12*c* adds a new capability for SQL plans called adaptive plan management. This feature lets you specify multiple plans for query optimization and have the optimizer select between them based on runtime statistics and user directives.

Comparing Optimizations

Oracle makes changes to the optimizer in every release. These changes are meant to improve the overall quality of the decisions the optimizer makes, but a generally improved optimizer could still create an execution plan for any particular SQL statement that could result in a decrease in performance.

The Oracle SQL Plan Analyzer tool is designed to give you the ability to recognize potential problems caused by optimizer upgrades. This tool compares the execution plans for the SQL statements in your application, flagging the ones in which the plans differ. Once these statements are identified, SQL Plan Analyzer executes the SQL in each environment and provides feedback on the performance and resource utilization for each. Although SQL Plan Analyzer cannot avoid potential problems brought on by optimizer upgrades, the tool can definitely simplify an otherwise complex testing task.

Oracle Database 11*g* also includes a feature called Database Replay. This feature captures workloads from production systems and allows them to be run on test systems. With this capability, you can test actual production scenarios against new configurations or versions of the database, and Database Replay will spot areas of potential performance problems on the changed platform. Both SQL Plan Analyzer and Database Replay are part of the Real Application Testing toolkit from Oracle.

Performance and Optimization

The purpose of the optimizer is to select the best execution plan for your queries. But there is a lot more to optimizing the overall performance of your database. Oracle performance is the subject of Chapter 7 of this book.

SQL Translation

The Oracle Database supports a rich set of SQL syntax; however, other databases also have their own SQL syntax, which may differ from the exact syntax supported in Oracle. In the past, these differences would require a sometimes burdensome change in the actual SQL statements in an application.

Oracle Database 12*c* introduces a SQL Translation Framework. This feature uses SQL Translators to intercept non-Oracle SQL statements and translate the statements into Oracle syntax. The SQL statements are captured and translated, and the translations are automatically used once they are created.

If a SQL statement cannot be translated, the Oracle Database returns an error. You can add custom translations using the same framework, allowing you to address SQL issues without having to change the actual SQL statements in the application.

You would use SQL Developer, a very popular tool for the Oracle Database, to implement not only SQL translation, but to migrate data structures and data from other databases. SQL Developer is available as a free download from the Oracle Technology Network, and includes a great deal of useful functionality for managing and administration of your Oracle Database and Oracle Database Cloud Service, discussed in Chapter 15.

Understanding the Execution Plan

Oracle's query optimizer uses an execution plan for each query submitted. By and large, although the optimizer does a good job of selecting the execution plan, there may be times when the performance of the database suggests that it is using a less-than-optimal execution plan.

The only way you can really tell what path is being selected by the optimizer is to see the layout of the execution plan. You can use two Oracle character-mode utilities to examine the execution plan chosen by the Oracle optimizer. These tools allow you to see the successive steps used by Oracle to collect, select, and return the data to the user.

The first utility is the SQL EXPLAIN PLAN statement. When you use EXPLAIN PLAN, followed by the keyword FOR and the SQL statement whose execution plan you want to view, the Oracle cost-based optimizer returns a description of the execution plan it will use for the SQL statement and inserts this description into a database table. You can

subsequently run a query on that table to get the execution plan, as shown in SQL*Plus in Figure 4-5.

```
SQL> EXPLAIN PLAN FOR
  2  SELECT DNAME, ENAME FROM EMP, DEPT
  3  WHERE EMP.DEPTNO = DEPT.DEPTNO
  4  ORDER BY DNAME;

Explained.

SQL> SELECT OBJECT_NAME, OPERATION, OPTIONS FROM PLAN_TABLE ORDER BY ID;

OBJECT_NAME                    OPERATION                      OPTIONS
------------------------------ ------------------------------ -------------------------
                               SELECT STATEMENT
                               SORT                           ORDER BY
                               NESTED LOOPS
EMP                            TABLE ACCESS                   FULL
DEPT                           TABLE ACCESS                   BY INDEX ROWID
SYS_C004911                    INDEX                          UNIQUE SCAN
6 rows selected.
```

*Figure 4-5. Results of a simple EXPLAIN PLAN statement in SQL*Plus*

The execution plan is presented as a series of rows in the table, one for each step taken by Oracle in the process of executing the SQL statement. The optimizer also includes some of the information related to its decisions, such as the overall cost of each step and some of the statistics that it used to make its decisions. You can also view an execution plan for a single SQL statement with SQL Developer or in the SQL Workshop area of Application Express, which is discussed in Chapter 15.

The optimizer writes all of this information to a table in the database. By default, the optimizer uses a table called PLAN_TABLE; make sure the table exists before you use EXPLAIN PLAN. (The *utlxplan.sql* script included with your Oracle Database creates the default PLAN_TABLE table.) You can specify that EXPLAIN PLAN uses a table other than PLAN_TABLE in the syntax of the statement. For more information about the use of EXPLAIN PLAN, please refer to your Oracle documentation.

There are times when you want to examine the execution plan for a single statement. In such cases, the EXPLAIN PLAN syntax is appropriate. There are other times when you want to look at the plans for a group of SQL statements. For these situations, you can set up a trace for the statements you want to examine and then use the second utility, TKPROF, to give you the results of the trace in a more readable format in a separate file. At other times, you might also use Oracle's SQL Trace facility to generate a file containing the SQL generated when using TKPROF in tuning applications.

You must use the EXPLAIN keyword when you start TKPROF, as this will instruct the utility to execute an EXPLAIN PLAN statement for each SQL statement in the trace file. You can also specify how the results delivered by TKPROF are sorted. For instance, you

can have the SQL statements sorted on the basis of the physical I/Os they used; the elapsed time spent on parsing, executing, or fetching the rows; or the total number of rows affected.

The TKPROF utility uses a trace file as its raw material. Trace files are created for individual sessions. You can start collecting a trace file either by running the target application with a switch (if it's written with an Oracle product such as Developer) or by explicitly turning it on with an EXEC SQL call or an ALTER SESSION SQL statement in an application written with a 3GL. The trace process, as you can probably guess, can significantly affect the performance of an application, so you should turn it on only when you have some specific diagnostic work to do.

You can also view the execution plan through Enterprise Manager for the SQL statements that use the most resources. Tuning your SQL statements isn't a trivial task, but with the EXPLAIN PLAN and TKPROF utilities you can get to the bottom of the decisions made by the cost-based optimizer. It takes a bit of practice to understand exactly how to read an execution plan, but it's better to have access to this type of information than not. In large-scale system development projects, it's quite common for developers to submit EXPLAIN PLANs for the SQL they're writing to a DBA as a formal step toward completing a form or report. While time-consuming, this is the best way to ensure that your SQL is tuned before going into production.

SQL Advisors

Oracle Database 10*g* added a tool called the SQL Tuning Advisor. This tool performs advanced optimization analysis on selected SQL statements, using workloads that have been automatically collected into the Automatic Workload Repository or that you have specified yourself. Once the optimization is done, the SQL Tuning Advisor makes recommendations, which could include updating statistics, adding indexes, or creating a SQL profile. This profile is stored in the database and is used as the optimization plan for future executions of the statement, which allows you to "fix" errant SQL plans without having to touch the underlying SQL.

The tool is often used along with the SQL Access Advisor since that tool provides advice on materialized views and indexes. Oracle Database 11*g* introduces a SQL Advisor tool that combines functions of the SQL Tuning Advisor and the SQL Access Advisor (and now includes a new Partition Advisor). The Partition Advisor component advises on how to partition tables, materialized views, and indexes in order to improve SQL performance.

Data Dictionary Tables

The main purpose of the Oracle data dictionary is to store data that describes the structure of the objects in the Oracle Database. Because of this purpose, there are many views

in the Oracle data dictionary that provide information about the attributes and composition of the data structures within the database.

All of the views listed in this section actually have three varieties, which are identified by their prefixes:

*DBA_*
: Includes all the objects in the database. A user must have DBA privileges to use this view.

*USER_*
: Includes only the objects in the user's own database schema.

*ALL_*
: Includes all the objects in the database to which a particular user has access. If a user has been granted rights to objects in another user's schema, these objects will appear in this view.

This means that, for instance, there are three views that relate to tables: DBA_TABLES, USER_TABLES, and ALL_TABLES. Oracle Database 12*c* includes views for container databases prefixed with *CDB_*.

Some of the more common views that directly relate to the data structures are described in Table 4-2.

Table 4-2. Data dictionary views about data structures

Data dictionary view	Type of information
ALL_TABLES	Information about the object and relational tables
TABLES	Information about the relational tables
XML_TABLES	Information about XML tables
TAB_COMMENTS	Comments about the table structures
TAB_HISTOGRAMS	Statistics about the use of tables
TAB_PARTITIONS	Information about the partitions in a partitioned table
TAB_PRIVS*	Different views detailing all the privileges on a table, the privileges granted by the user, and the privileges granted to the user
TAB_COLUMNS	Information about the columns in tables and views
COL_COMMENTS	Comments about individual columns
COL_PRIVS*	Different views detailing all the privileges on a column, the privileges granted by the user, and the privileges granted to the user
LOBS	Information about large object (LOB) datatype columns
VIEWS	Information about views
INDEXES	Information about the indexes on tables
IND_COLUMNS	Information about the columns in each index
IND_PARTITIONS	Information about each partition in a partitioned index

Data dictionary view	Type of information
PART_*	Different views detailing the composition and usage patterns for partitioned tables and indexes
CONS_COLUMNS	Information about the columns in each constraint
CONSTRAINTS	Information about constraints on tables
SEQUENCES	Information about sequence objects
SYNONYMS	Information about synonyms
TAB_COL_STATISTICS	Statistics used by the cost-based analyzer
TRIGGERS	Information about the triggers on tables
TRIGGER_COLS	Information about the columns in triggers

CHAPTER 5
Managing Oracle

Many Oracle users are not fully aware of system and Oracle Database management activities that go on around them. But effective database and infrastructure management is vital to providing a reliable, available, and secure platform that delivers optimal performance. This chapter focuses on how Oracle can be managed to ensure these virtues for your environment.

Much of the management responsibility usually falls upon the database administrator (DBA). The DBA is typically responsible for the following management tasks:

- Installing and patching the database and options
- Creating tables and indexes
- Creating and managing tablespaces
- Managing control files, online redo logs, archived redo logs, job queues, and server processes
- Creating, monitoring, and tuning data-loading procedures
- Adding users and roles and implementing security procedures
- Implementing backup, recovery, information lifecycle management, and high availability plans
- Monitoring database performance and exceptions
- Reorganizing and tuning the database
- Troubleshooting database problems
- Coordinating with Oracle Global Customer Support

Where Oracle's engineered systems such as the Oracle Exadata Database Machine are deployed, DBAs often take on other responsibilities such as operating system monitoring and patching, management of storage, and hardware monitoring and troubleshooting.

Particularly in smaller companies, DBAs are often called upon to take part in database schema design and security planning. DBAs in large enterprises may also help set up replication strategies, disaster and high-availability strategies, hierarchical storage management procedures, and the linking of database event monitoring (e.g., specific database tasks and status) into enterprise network monitors.

The feature list has grown with each Oracle Database release—inevitably, since more flexible functionality means more to configure and monitor. Yet managing Oracle can be much less labor-intensive today than it was in the past. While early editions of this book described the novelty of an easier-to-use management interface for Oracle, producing better versions of Oracle Enterprise Manager (EM) was only part of the effort to simplify management underway within Oracle Server Development. The database itself has become more self-tuning and self-managing with each release.

Initially, this effort was focused mostly on better management of single instances of the Oracle Database. For Oracle Database 10*g* and Oracle Database 11*g*, the focus grew to effective management of scores of computers and Oracle Database instances and clusters common in grid computing. For Oracle Database 12*c*, there is added emphasis on managing and provisioning Oracle Databases deployed on public and private clouds.

Manageability of grid and cloud infrastructures must take into account disk virtualization, resource pooling, provisioning of computer resources, dynamic workload management, and dynamic control of changing components. Oracle's initiatives in these areas resulted in many significant changes in managing the database geared toward significantly reducing this complexity. While targeted at simplifying complex enterprise management tasks where dozens or hundreds of Oracle Databases are deployed, most of these improvements also have a significant impact in simplifying management of more traditional Oracle Database implementations.

As a consequence of the grid, cloud, and self-tuning and self-managing initiatives, readers of early editions of this book will find a large number of management changes in this chapter and in other related chapters throughout this book.

All of the tasks we've just described come under the heading of managing the database. Many of the provisioning duties, including installation and initial are discussed in Chapter 3. Security issues are discussed in Chapter 6. This chapter explores the following aspects of managing Oracle:

- Database manageability features and advisors and how they aid management
- Oracle Enterprise Manager, which provides an intuitive interface and underlying framework for many database management tasks
- Backup and recovery operations and information lifecycle management, which are the foundation of database integrity protection
- Oracle Support

In subsequent chapters, we'll cover other related topics in more depth, including security, performance, and high availability. You will need an understanding of all of these areas as you plan and implement effective management strategies for your Oracle Database environment.

Manageability Features

Oracle's goal of simplifying management of the database became clear with the introduction of the "Intelligent Infrastructure," first highlighted with Oracle Database 10*g*, eliminating many manual steps required to manage the database. Subsequent database versions and releases added more self-tuning and self-management features and advisor tools. Automatic database optimizer statistics gathering, the Segment Advisor, and the SQL Tuning Advisor are just a few of the many such features introduced.

Today, statistics containing active session history are automatically gathered and populate the Automatic Workload Repository (AWR). The Automatic Database Diagnostic Monitor (ADDM) tracks changes in database performance by leveraging the data in the AWR. Server-generated alerts occur "just in time" and appear in Enterprise Manager. Resolving system utilization problems can be as simple as reviewing the alerts and accepting the recommendations. This is in sharp contrast to steps typically taken prior to Oracle Database 10*g* that included actively watching for events, exploring `V$` views, identifying related SQL, and then figuring out the needed steps to resolve the problem.

With its focus on the cloud, Oracle Database 12*c* introduces pluggable databases that can further simplify management where large numbers of Oracle Databases are deployed since many management functions are more efficient when defined for multitenant container databases instead of dozens of pluggable databases. The number of database management activities online also continues to grow, further simplifying management considerations (e.g., datafile movement, partition and subpartition movement, and redefinition of tables that contain multiple partitions or that have Virtual Private Database (VPD) policies are a few of the recent additions).

Key to understanding the infrastructure Oracle has created for managing the database is exploring the various management components. We will start with the Oracle Database advisors and then describe the role of Automatic Storage Management.

Database Advisors

Oracle's advisors deliver database performance, recoverability, and diagnostic service improvement recommendations via Oracle Enterprise Manager. Each database advisor has a unique role. Many can be set to automatically take action if preferred.

The Automated Database Diagnostic Monitor (ADDM) performs real-time database performance diagnostics, recommends potential solutions to performance problems, and quantifies benefits of those solutions. ADDM, usually set to run every hour, monitors current snapshots of the database state and workload gathered in the AWR. ADDM can also be manually triggered to analyze historical data for specific time periods. Real-time ADDM detects and diagnoses transient problems that include high CPU loads, I/O in CPU bound situations, over allocated memory, interconnect issues, session and process limits being reached, and hangs and deadlocks. In Oracle Database 12*c*, real-time ADDM is automatically run every three seconds, enabling DBAs to resolve deadlocks, hangs, shared pool connection issues, and other similar situations without having to restart the Oracle Database.

Other Oracle Database performance-related advisors include:

SQL Tuning Advisor
: Available for the Oracle Database since Oracle Database 11*g*, the SQL Tuning Advisor analyzes SQL statements, object statistics, access paths, and degree of parallelism to make improvement recommendations such as creating a new SQL profile or choosing an alternate execution plan from the AWR. In automatic mode, it will automatically create a better plan if that plan will yield significant improvement.

SQL Access Advisor
: Also available since Oracle Database 11*g*, the SQL Access Advisor provides recommendations on materialized views, indexes, and materialized view logs that should be created, dropped, and retained. The Advisor also makes recommendations about partitioning to improve performance.

SPM Evolve Advisor
: New in Oracle Database 12*c*, the SQL Plan Management (SPM) Evolve Advisor enables scheduling of testing of new plans added to the SQL plan baseline, described in Chapter 4, compares the new plans versus accepted plans for cost, and automatically accepts plans that have a much lower cost.

Memory Advisors
: The Memory Advisor is an expert system that provides automatic memory management and eliminates manual adjustment of the SGA and PGA when enabled (and recommended in Oracle Database 11*g* or more recent releases). If just automatic shared memory is enabled instead, you will have access to the Shared Pool (SGA) Advisor and PGA Advisor. Finally, if you are manually managing shared

memory, you will have access to the Shared Pool (SGA) Advisor, Buffer Cache Advisor, and PGA Advisor.

Java Pool Advisor
: The Java Pool Advisor provides statistics about library cache memory used for Java and predicts parse rate based on changes in the Java pool size.

Segment Advisor
: Use of the Segment Advisor eliminates the need to identify fragmented objects and reorganize the objects using scripts. The Segment Advisor advises which objects to shrink online. For tables that cannot be shrunk, it will recommend an online table redefinition. It can also make OLTP compression recommendations. The Segment Advisor can be run automatically or manually. (Note that Automatic Segment Space Management or ASSM is the default for locally managed tablespaces in recent Oracle Database releases.)

Advisors can also provide recommendations as you determine the recoverability characteristics of the Oracle Database. Such advisors include:

Undo Advisor
: The Undo Advisor helps size a fixed-size undo tablespace and can be used to set the low threshold of undo retention for Flashback capabilities. You first estimate the length of your longest running query and the longest interval you will need for Flashback. The larger of these two values then serves as input to the Undo Advisor which returns a size recommendation.

MTTR Advisor
: The Mean Time to Recovery (MTTR) Advisor provides guidance regarding the impact of MTTR settings and physical writes. The mean time for recovery from a system failure is specified based on business needs by the database administrator using Enterprise Manager, and then reconfiguration of Oracle parameters takes place to match requirements.

Another class of advisors used to resolve database and SQL issues first became available with Oracle Database 11*g*. When critical errors are detected, the fault diagnosis infrastructure for the Oracle Database can perform a deeper analysis called a health check using a Health Monitor. The advisors leverage diagnostic data including database traces, the alert log, Health Monitor reports, and other diagnostic information stored in the Automatic Diagnostic Repository (ADR). The infrastructure also includes a SQL Test Case Builder used for gathering information about problems and transmitting the information to Oracle Support. The advisors in this infrastructure include:

SQL Repair Advisor
: If a SQL statement fails with a critical error, the SQL Repair Advisor will analyze the statement and recommend a patch to fix it.

Data Recovery Advisor
: The Data Recovery Advisor is used in recovering from corrupted blocks, corrupted or missing files, and other data failures and is integrated with database health checks and RMAN.

Automatic Storage Management

Part of the Oracle Database since Oracle Database 10*g*, Automatic Storage Management (ASM) provides a file system and volume manager in the database, enabling automated striping of files and automating mirroring of database extents. DBAs simply define a pool of storage or disk group and manage the disk group through EM. Disk groups are created with normal redundancy as the default (two-way mirroring). You can also create disk groups with high redundancy (three-way mirroring) or external redundancy (no mirroring). Failure groups are ASM disks that share a common failure point, so mirroring will automatically occur to a different failure group to provide high availability.

Oracle manages the files that are stored in ASM disk groups. ASM manages Oracle datafiles, logfiles, control files, archive logs, and RMAN/backup sets. Workloads can be dynamically rebalanced as storage is reconfigured such that when storage is added or removed from the pool, data can be redistributed in the background.

Oracle Database 12*c* introduces a number of features that improve the availability and reliability of ASM. ASM disk scrubbing provides automatic repair of logical data corruptions. ASM disk resync allows multiple disks to be brought back online simultaneously.

Oracle Enterprise Manager

Oracle Enterprise Manager was first distributed with Oracle7 and was focused at that time on just simplifying database management. Early EM versions required Windows-based workstations as client machines. A Java applet browser-based EM console appeared with the Oracle8*i* database release. The HTML-based console was introduced with Oracle9*i* and is now the basis for Enterprise Manager 12*c* used to manage the database and many other Oracle products and platforms for cloud-based and other deployments. Alternatively, Enterprise Manager 12*c* can be accessed via iOS devices including iPhones and iPads.

Oracle Database 12*c* also comes with a small footprint version of Enterprise Manager called Enterprise Manager Express that requires no additional middleware. Enterprise Manager Express provides basic storage, security, and configuration administration support as well as advanced performance diagnostics and tuning.

Enterprise Manager provides a basic interface for monitoring and management of Oracle Database users and user privileges, database schema and database configuration

status, and backup and recovery. Today, EM is far more than just an Oracle Database management interface and can be used for the following:

Database management
: Enables DBA to more easily perform database change and configuration management, patching, provisioning, tasking, masking and subsetting, and performance management and tuning while leveraging automated management capabilities in the Oracle Database

Database lifecycle management
: Oracle Database discovery, initial provisioning, patching, configuration management, and ongoing change management

Exadata management
: Provides unified view of Exadata nodes, Storage Server cells, InfiniBand switches, software running on them, and resource utilization

Hardware and virtualization management
: Used for managing Oracle VM where Linux, Unix/Solaris, and Windows are deployed and integrated with the Oracle Virtual Assembly Builder

Middleware management
: Configuration and lifecycle management for Oracle WebLogic Server, SOA Suite, Coherence, Identity Management, WebCenter, Oracle Business Intelligence, and the Exalogic Elastic Cloud

Cloud management
: Cloud management for the Oracle Database and Oracle Fusion Middleware that includes self-service provisioning balanced against centralized policy-based resource management, integrated chargeback, and capacity planning

Heterogeneous (non-Oracle) management
: Oracle and partner plug-ins and connectors posted in the Enterprise Manager 12*c* Extensibility Exchange that enable Enterprise Manager to also manage TimesTen, Amazon Web Services, IBM DB2, IBM Websphere Application Server, Sun ZFS Appliance, EMC CLARiiON, and many others

Packaged applications management
: Application monitoring and management of Oracle applications (E-Business Suite, PeopleSoft, Siebel, and Fusion applications)

Application performance management
: Real user monitoring and synthetic transaction monitoring, monitoring and tracing of transactions/transaction instances, Java and Oracle Database monitoring and diagnostics, multilayer discovery of application topology and infrastructure, and application performance analytics and reporting for Oracle applications and custom applications

Application quality management
: Application test management, functional testing, load testing, Real Application Testing (RAT), and test data management for Oracle applications and custom web-based applications running on the Oracle Database

Since the focus of this book is on the Oracle Database, we'll focus the remainder of this chapter on capabilities in EM for managing the database infrastructure. For database management, in addition to basic management capabilities provided in EM, Oracle offers a number of optional packs, including:

Diagnostic Pack for Oracle Database
: Provides automatic performance diagnostics, flexible system monitoring and notification, extended Exadata management, and database problem root cause analysis and baseline comparison interfaces, leveraging ADDM and the AWR in the Oracle Database

Tuning Pack for Oracle Database
: Provides a real-time SQL monitor useful in identifying long-running queries, the SQL Tuning Advisor for analysis and automatic implementation of improved SQL profile recommendations, the SQL Access Advisor for recommending schema redesign, and an object reorganization wizard for removing wasted space

Lifecycle Management Pack for Oracle Database
: Provides database discovery and inventory tracking, initial database provisioning, ongoing change management of patches, upgrades, and schema and data changes, configuration and compliance management, and site-level disaster protection automation (also called Site Guard)

Oracle Data Masking Pack
: Provides search for sensitive data capability, common data masking formats, and supports condition-based masking, compound masking, deterministic masking, and key-based reversible masking

Oracle Test Data Management Pack
: Detects and stores data relationships from a production Oracle Database in an Application Data Model which is then used to generate a subset of data useful in nonproduction Oracle Database testing

Cloud Management Pack for Oracle Database
: Provides a Consolidation Planner for determining how to redistribute workloads and meet service level agreements (SLAs), a self-service portal through which new instances can be requested for single instance and Real Application Clusters (RAC) Oracle Databases and then started and monitored, and also provides a metering and chargeback interface

What About Database Fragmentation?

In earlier editions of *Oracle Essentials*, one area we focused on in this chapter was database fragmentation, where small amounts of "free space" on disk could negatively impact database performance. Remember that Oracle consists of blocks called extents. A collection of extents is referred to as a segment, and segments contain anything that takes up space—for example, a table, an index, or a rollback segment. Segments typically consist of multiple extents. As an extent fills up, a segment begins to use another extent. When fragmentation occurs by database activities that leave "holes" in the continuous space represented by extents, segments acquire additional extents. As fragmentation grows, increased I/O activity results in reduced database performance.

Prior to Oracle Database 10*g*, a popular topic at almost every Oracle Users Group conference was how to manage this fragmentation. Since Oracle Database 10*g*, you can simply perform an online segment shrink using the Tuning Pack (and Segment Advisor) noted above. ADDM recommends the segments to shrink, and you simply choose to accept the recommendations. You might also deploy locally managed tablespaces defaulted to ASSM.

In addition to solving fragmentation issues, Oracle has been working to diminish the impact of all disk I/O activity, which has been the key performance bottleneck for years. With Smart Scans and Smart Flash Cache, as well as ASM and many disks, Oracle Exadata engineered systems have gone a long way towards eliminating problems stemming from I/O access times.

Enterprise Manager Architecture

Enterprise Manager can be used for managing the database locally, remotely, and/or through firewalls. Individual consoles can manage single or multiple Oracle Databases. Since EM is used in managing Oracle deployed to on-premise servers or as part of cloud deployment, it is now often referenced as *Cloud Control*.

The Cloud Control console is your interface into EM, showing the software being managed and providing a high-level view of the status of components. You can drill from Cloud Control into the consoles for individual databases, application servers, and other targets. Figure 5-1 illustrates a typical Cloud Control home page.

EM also supports a command line interface (EMCLI).

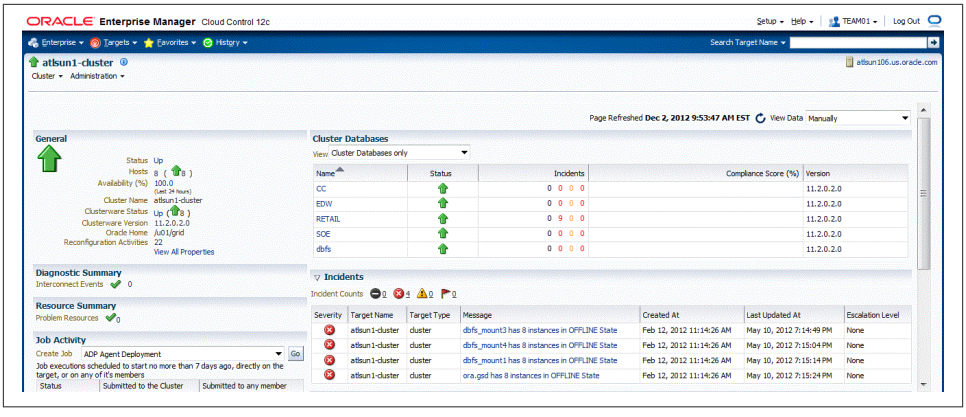


Figure 5-1. Typical Cloud Control home page

Other key Enterprise Manager components in its architecture include the following:

Oracle Management Agents
: Deployed on each host/database instance to be monitored, the agents automatically discover all Oracle components. The agents track each target's health, status, and performance and also relevant hardware and software configuration data and store the data locally in XML files prior to uploading the XML files. Management Agents can also perform predefined jobs and can be used to send Simple Network Management Protocol (SNMP) traps to database performance monitors required by other system monitoring tools. Management Agents are available for the wide variety of operating systems on which the Oracle Database is available.

Oracle Management Service (OMS)
: The OMS is a middle-tier web application (requiring the WebLogic Server) that, working with Oracle Management Agents, discovers targets, monitors and manages the targets, and stores the information it collects in the Oracle Management Repository by synchronously uploading XML files generated by the agents. OMS also renders the Cloud Control console user interface. To avoid EM outages, Oracle recommends deployment of multiple OMSs with a server load balancer.

Oracle Management Repository
: The Management Repository is configured in an Oracle Database and stores performance and availability data and configuration and compliance data uploaded by the OMS from Management Agents on the managed targets. Once in the repository, the data can be accessed by designated administrators through Enterprise Manager Cloud Control. The Management Repository also stores EM configuration information including users and privileges, monitoring settings, and job definitions. Where OMS outages are to be avoided, Oracle recommends deployment of the Management Repository in a RAC configuration.

A simple EM architecture diagram is shown in Figure 5-2.

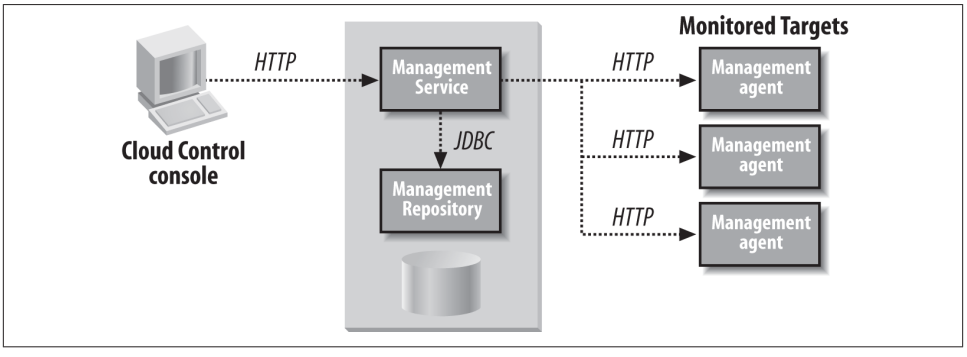


Figure 5-2. Oracle Enterprise Manager architecture

Oracle Management Agent and OMS software updates are provided through Oracle Management plug-ins on a more frequent basis than updates to EM itself. During a new EM installation, by default Oracle Management plug-ins are installed for the Oracle Database, Fusion Middleware, My Oracle Support, and Oracle Exadata. Plug-in updates can be automatically downloaded to EM by setting up the Self Update console.

EM supports three different categories of administrators. SYSMAN is in the most powerful category, the Super Administrator, and is created when EM is installed. Super Administrators have full access privileges to all targets and administrator accounts. They also create other types of administrators and have unique management capabilities. For example, Super Administrators can create OS-based notifications, set up notifications based on SNMP traps, troubleshoot the Management Repository, set up Enterprise Manager for self update, and fully manage the Software Library (where patches, virtual appliance images, reference gold images, application software, and directive scripts are stored).

More commonly, administrators are classified as "regular" administrators. The Super Administrator usually divides up work by granting access to regular administrators to certain targets and/or by giving them certain management task capabilities. A third type of EM administrator is the Repository Owner who is the DBA for the Management Repository.

A key underpinning to Enterprise Manager 12*c* used in managing Oracle hardware servers and virtual machines is Enterprise Manager Ops Center 12*c*. Ops Center 12*c* manages Oracle servers that support the Integrated Lights Out Manager (ILOM), Advanced Lights Out Manager (ALOM), and Embedded Lights Out Manager (ELOM). These include Oracle's engineered systems, M-class servers, ZFS Storage Appliances, and InfiniBand and Ethernet switches. Supported operating systems include Oracle Solaris, Oracle Linux, Red Hat, SuSE, and Windows. Management capabilities extend

to Oracle VM and Oracle Solaris Containers. Among the supported features are discovery and topology management, support for virtual DataCenter and server pool architectures, fault monitoring, Automatic Service Requests (ASR) and My Oracle Support integration, BIOS and firmware automation, operating system bare metal provisioning and performance monitoring, patch automation, configuration and compliance reporting, and energy usage awareness and management.

Oracle Enterprise Manager Consoles

EM's popularity grew as deployment of the Oracle Database expanded within companies to multiple operating systems and as additional Oracle software components were added to the mix. EM provides a common interface to managing all of these environments in a wide array of traditional and cloud deployment models, something that DBA scripts were not usually designed for. Further, the Enterprise Manager 12*c* Cloud Control console and framework provide simple access to new database self-monitoring features, respond to alerts, and manage jobs, reports, roles, and privileges.

We are now going to look further at the layout of the Enterprise Manager 12*c* Cloud Control console, current as this edition of *Oracle Essentials* was published. Oracle often changes the scope of Enterprise Manager capabilities and interface details, so you will want to explore the version you have deployed. But some of the basic functionality for database management consistently appears in the various EM versions.

Logging into Enterprise Manager after installation requires that you provide an administrator username and password. As we saw in Figure 5-1, when viewing databases you are monitoring, you are presented with the general status, a diagnostic and resource summary, job activities, the status of individual Oracle Databases, an incident report, and a list of hosts and their status (not shown in the figure due to limitations in how much is visible on the screen—you'd simply scroll down the screen to see this). You'll also notice in the upper-left corner of the screenshot links into cluster management and administration.

Cluster management provides monitoring metrics concerning the cluster status history, an incident manager, alert history, and list of blackouts. Job activities in the cluster are presented and there is an interface for publishing standard reports about the overall status. There is also a performance status area that shows performance of interconnects and member targets. The cluster topology and compliance status can be viewed here (see Figure 5-3). There is also an interface for setting up new targets.

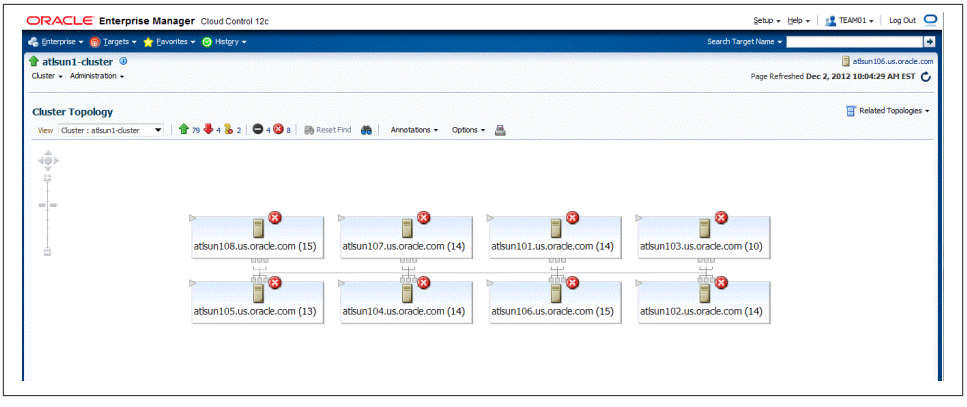


Figure 5-3. Oracle Enterprise Manager 12c cluster topology diagram

As we explore the main administration link, we first see a page that gives us summaries of database compliance (see Chapter 6), status, and diagnostics. This page also provides a summary of performance, status of key database resources (host CPU, active sessions, memory allocation, and data storage), SQL Monitor, instances status, and a list of incidences and problems. We see a portion of this page in Figure 5-4. In the upper left are links that provide access to performance, availability, schema, and other administration management tools.

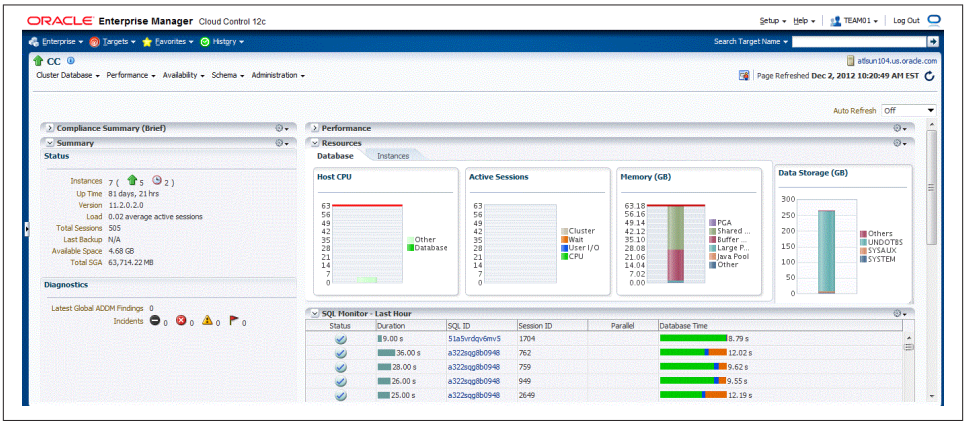


Figure 5-4. Oracle Enterprise Manager 12c database administration overview

Let's now take a look at where interfaces to the database management tools are presented:

Performance page
: Includes Top Activity view, SQL Monitoring, Cluster Cache Coherency status, SQL Tuning Advisor, Performance Analyzer, SQL Access Advisor, other SQL tools and

worksheets, links to AWR and the home for all Advisors, ADDM and Emergency Monitoring interface, Sessions status, and Database Replay

Availability page
: Exposes the High Availability console, Maximum Availability Architecture (MAA) Advisor, Backup and Recovery interface, Add Standby Database interface, and Cluster Managed Services and Operations interface

Schema page
: Includes links to interfaces for managing database users and privileges, Database objects (tables, indexes, views, synonyms, sequences, and database links), database programs, materialized views, user-defined types, database export and import, database change management, data discovery and modeling, data subsetting, data masking, XML DB, Text Manager, and workspace management

Administration page
: Includes interfaces to manage database initialization parameters, security, storage, the Oracle Scheduler, Streams and replication, Exadata, and ASM (that includes migrate to home, diskgroups, Database Resource Manager, and database feature usage).

Figure 5-5 illustrates the interface into management of Exadata.

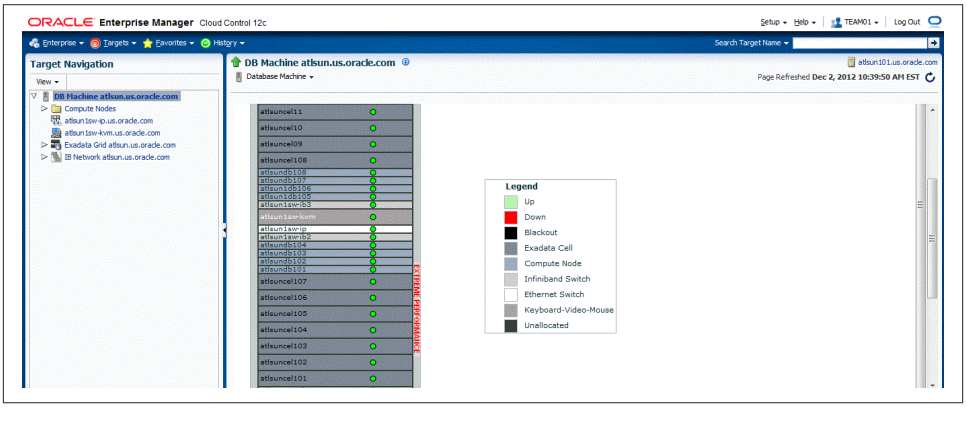


Figure 5-5. Oracle Enterprise Manager 12c Exadata management view

You can access the full Enterprise Manager 12*c* Cloud Control console through a browser on your mobile device. There is also an Enterprise Manager Cloud Control Mobile application available for installation on iOS-based mobile devices. The Cloud Control Mobile application is downloadable from the Apple iTunes App Store. The Cloud Control Mobile interface accesses the URL of your Enterprise Manager 12*c* Cloud Control console via WiFi or a mobile network connection over VPN. The URL is typically of the form *https://youroraclesiteid.com/em*. The Cloud Control Mobile interface provides

a means to view incidents and problems in detail, acknowledge an incident or problem, log into My Oracle Support and view an SR and updates related to a problem, and take actions via a manage dialog.

EM Express

As previously noted, Oracle Database 12*c* includes EM Express that is pre-configured, installed, and requires no middleware components as all of the rendering occurs in the browser. As you might expect, EM Express has more limited basic administration support for configuration, security, and storage, though it does support advanced performance diagnostics and tuning. Configuration management options include initialization parameters, memory, database feature usage, and current database properties. Storage management includes tablespaces, undo management, redo log groups, archive logs, and control files. Security management includes users, roles, and profiles. Performance management includes the performance hub and SQL Tuning Advisor.

Backup and Recovery

Even if you've taken adequate precautions, critical database records can sometimes be destroyed as a result of human error or hardware or software failure. The only way to prepare for this type of potentially disastrous situation is to perform regular backup operations. Ensuring reliable backup and recovery is a key part of defining a Maximum Availability Architecture (MAA) for your Oracle Databases that is further described elsewhere in this book.

Two basic types of potential failures can affect an oracle database: *instance failure*, in which the Oracle instance terminates without going through the shutdown process; and *media failure*, in which the disks that store the information in an Oracle Database are corrupted or damaged.

After an instance failure, Oracle will automatically perform crash recovery. For example, you can use Real Application Clusters to automatically perform instance recovery when one of its instances crashes. However, DBAs must initiate recovery from media failure. The ability to recover successfully from this type of failure will be the result of careful planning. The recovery process restores older copies of the damaged Oracle datafile(s) and rolls forward by applying archived and online redo logs.

To ensure successful recovery, the DBA should have prepared for this eventuality by performing the following actions:

- Multiplexing online redo logs by having multiple log members per group on different disks and controllers
- Running the database in ARCHIVELOG mode when possible so that redo logfiles are archived

- Archiving redo logs to multiple locations
- Maintaining multiple copies of the control file(s)
- Backing up physical datafiles frequently—ideally, storing multiple copies in multiple locations

Running the database in ARCHIVELOG mode enables the DBA to perform online datafile backups while the database is available for use since archived logs are available for the recovery process. The archived redo logs can also be sent to a standby database to which they may be applied. If the database is being run in NOARCHIVELOG mode, it must first be shut down and then mounted to perform a consistent backup and is not available during this time.

Types of Backup and Recovery Options

There are two major categories of backup:

Full backup
: Includes backups of Oracle datafiles, datafile copies, tablespaces, control files (current or backup), or the entire database (including all datafiles and the current control file) and reads entire files and copies all blocks into the backup set, skipping only datafile blocks that have never been used (with the exception of control files and redo logs where no blocks are skipped).

Incremental backup
: Can include backups of Oracle datafiles, tablespaces, or the entire database. Reads entire files and backs up only those data blocks that have changed since a previous backup.

You can begin backups through the Recovery Manager (RMAN) or the Oracle Enterprise Manager interface to RMAN, which uses the database export facility, or you can initiate backups via standard operating system backup utilities.

RMAN is designed to support the types of database backups you are likely to perform. These include open or online backups, closed database backups, incremental backups at the Oracle block level, corrupt block detection, automatic backups, backup catalogs, and backups to sequential media. Backups and restores can be restarted. The testing of restores and recovery is supported. By setting recovery windows, expiration dates of backups are determined and managed.

RMAN can perform image copy backups of the database, tablespaces, or datafiles. RMAN can be used to apply incremental backups to datafile image backups. The speed of incremental backups is increased by a change-tracking feature that reads and backs up only changed blocks.

Recovery options include the following:

- Complete database recovery to the point of failure
- Time-based or point-in-time database recovery (recovery of the entire database to a time before the most current time)
- Tablespace point-in-time recovery (recovery of a tablespace to a time different from the rest of the database)
- Recovery of individual tables (a new feature in Oracle Database 12*c*)
- Backup and recovery of multitenant container databases including backup and point-in-time recovery of individual pluggable databases (new in Oracle Database 12*c*)
- Recovery until the `CANCEL` command is issued
- Change-based or log sequence recovery (to a specified System Change Number, or SCN)

You can recover through RMAN, using either the recovery catalog or control file or via SQL or SQL*Plus.

RMAN also supports the backup and restore of standby control files, has the ability to automatically retry a failed backup or restore operation, and can automatically create and recover Oracle datafiles not in the most recent backup. Where backups are missing or corrupt during the restore process, RMAN automatically uses an older backup.

To speed backups and restore operations, the Flash Recovery Area, featured in the Oracle Database since Oracle Database 10*g*, organizes recovery files to a specific area on disk. These files include a copy of the control file, archived logfiles, flashback database logs, datafile copies, and RMAN backups. You can set a RETENTION AREA parameter to retain needed recovery files for specific time windows. As backup files and archive logs age beyond the time window, they are automatically deleted. ASM (described earlier in this chapter) can configure the Flash Recovery Area. If availability of disk space is an issue, you can also take advantage of RMAN's ability to compress backup sets.

Making Sure the Backup Works

The key to providing an adequate backup and recovery strategy is to simulate recovery from failure using the backups with your test system before using the backups to restore a live production database. Many times, backup media that were thought to be reliable prove not to be, or backup frequencies that were thought to be adequate prove to be too infrequent to allow for timely recoveries. It's far better to discover that recovery is slow or impossible in test situations than after your business has been affected by the failure of a production system.

This section provided only a very brief overview of standard backup and recovery. For more information on high availability and MAA, refer to Chapter 11.

Oracle Secure Backup

Oracle Secure Backup (OSB) for tape devices first appeared with Oracle Database 10*g* and serves as a tape media manager, leveraging RMAN's ability to read the database block layout directly. It supports major tape drives and tape libraries in SAN, gigabit Ethernet, and SCSI configurations. OSB can provide tape data protection for one server attached to a tape drive or support multiple drives for any number of servers. As a backup solution, it is unique in its ability to provide an interface for making RMAN encrypted backups directly to tape. The addition of the OSB Cloud Module enables backup to the Amazon S3 Cloud storage devices.

The architecture of OSB includes an administrative server that contains the backup catalog, configuration data, and performs as the certificate authority for server authentication, media servers that transfer data to and from tape devices, and clients that are the hosts to be backed up. OSB balances loads across like network interfaces from the client hosts to the media server hosts. Retention policies can be time-managed where administrators define when tape expiration will occur or they can be managed by RMAN.

Information Lifecycle Management

Oracle Information Lifecycle Management (ILM) for the Oracle Database and Oracle's ILM Assistant were introduced in 2006. The ILM Assistant provides a means to define classes of data, create storage tiers for the data classes, create data access and data movement policies, and implement data compliance policies. ILM is most frequently used to move data in a partitioned Oracle Database among various storage devices that are most appropriate for hosting that data. The reason for doing this is that most administrators would like to have their most frequently accessed data on the fastest but most expensive storage devices, and the least frequently accessed data on the slowest but cheapest storage. One of the common use cases for ILM is setting up an online archive.

The ILM Assistant presents a graphical user interface used in creating lifecycle definitions and policies for database tables. It can advise when it is time to move, archive, or delete data, and also illustrate cost savings and storage required. The ILM Assistant can also guide you in creating partitioning to match your ILM needs. Once you have defined a strategy, it generates the scripts for moving the data. In addition to the ILM Assistant, you will need to have Oracle Application Express installed in the database where the data is managed.

The ILM Assistant is also used to define logical storage tiers (the level of performance of the storage in that tier and preferred database tablespaces), define the data lifecycle (how data will migrate over time), generate partition advice if the table is not partitioned,

and assign tables to be managed by the Assistant. You might also view partition simulation, a lifecycle summary, and storage costs, and define policy notes during setup of the lifecycle.

ILM in Oracle Database 12*c*

Oracle Database 12*c* introduces *Heat Maps* and *Automatic Data Optimization* (ADO) for implementing ILM strategies. Oracle Database 12*c* tracks data access at the segment level in the Heat Maps. Data modification is tracked using Heat Maps at the segment and row level.

ADO provides the ability to automate the compression and movement of data among different tiers of storage in the database. Heat Maps must be enabled to use this feature. You can specify compression levels in the tiers and when data movement should take place based on data access activity levels. Storage tier policies are usually set at the segment level, though compression policies can also be set at the row level in combination with segment level policies for finer control. For example, you might specify a policy for compression of rows of data not modified in the past 45 days.

ADO is deeply integrated with the Oracle Database and, as with other ILM strategies, disk savings and performance benefits can be significant. It is expected that organizations will increasingly use the new automated capabilities as they deploy Oracle Database 12*c* and upgrade from older Oracle releases where the ILM Assistant was used.

Working with Oracle Support

Regardless of the extent of your training, there are bound to be some issues that you can't resolve without help from Oracle. Part of the job of the DBA is to help resolve any issues with the Oracle Database. Oracle offers several levels of support, including Premier Support, Extended Support, and Sustaining Support. Given how critical the Oracle Database is to the business, most organizations choose Premier Support.

Premier Support includes software updates, fixes, security alerts, critical patch updates, and upgrade scripts. You have access to the web-based My Oracle Support system 24 hours a day and 7 days a week, including the ability to log Service Requests (SRs) online. You also can get assistance with SRs 24/7.

If you have Premier Support and have deployed the Oracle Database on an Oracle-engineered system such as the Exadata Database Machine, you have access to Platinum Services at no extra cost. Oracle provides remote fault monitoring of the system and Oracle Database 24/7 with response times of 5 minutes for fault notification, 15 minutes for restoration or escalation to Oracle Development, and 30 minutes for joint debugging with Oracle Development to begin. Full Stack Download Patches (QDPs) are provided four times a year.

Oracle's Advanced Customer Support (ACS) organization provides additional Oracle Database support through optional services. ACS Services offered include standard system installation, standard software installation and configuration, pre-production readiness review, production diagnostic review and recommendations, and patch review and installation. Oracle Consulting and Oracle partners also provide additional customization services.

Reporting Problems

The most common way of reporting SRs is via the web browser-based My Oracle Support interface. My Oracle Support has grown to be extremely popular, since you might find posted answers to problems similar to yours, eliminating time required for a physical response. My Oracle Support also provides proactive notifications and access to technical libraries and forums, product lifecycle information, and a bug database.

When contacting technical support, you will need your Customer Support Identification (CSI) number. Oracle Sales Consultants can also provide advice regarding how to report problems and Oracle ACS offers workshops for DBAs regarding more effectively supporting the database. As you report your SR, it is important for you to choose the right severity level. The severity levels are defined by Oracle as follows:

Severity 1
: Your system hangs indefinitely or crashes repeatedly after restarts, data is corrupted, or a critical documented function necessary to run your business is not available

Severity 2
: A severe loss of service occurs with important features not available though business operations can continue in a restricted fashion

Severity 3
: A minor loss of service occurs; however, a workaround is available

Severity 4
: You've requested information or an enhancement where no loss of service is involved

If business is halted because of the problem, the severity level assigned should be "severity 1." However, if a problem is reported at level 1, you must be available to work with Oracle Support (even if after hours). Otherwise, Oracle will assume that the problem wasn't as severe as initially reported and may lower the priority level for resolution.

A Support Workbench is accessible in Enterprise Manager through the Incident Manager and is used to package diagnostics from the ADR and open an SR. A SQL Test Case Builder automates the gathering of information about the problem and environment for uploading to Oracle Support to help them re-create the problem and resolve it sooner. You typically would first see problems as critical error alerts on the Enterprise

Manager Database Cloud Control console, then would view problem details, gather additional information, create an SR, package and upload the diagnostic data to Oracle Support, then track the SR and close it when resolved.

Automated Patching

Oracle supports automated patching of the Oracle Database in a couple different ways. The recommended way is to use Enterprise Manager 12*c* Cloud Control and leverage the agents present on each managed target to collect configuration data. From the Cloud Control console, you can select recommended patches, add to your plans once you've validated them, and deploy to selected targets. Once applied, they will disappear from the recommended patch list.

Alternatively, you can install the Oracle Configuration Manager or Enterprise Manager Cloud Control for My Oracle Support configuration collection on each managed target. You can then add recommended patches to plans once you have validated them, but the deployment process is not automated beyond this.

In RAC environments, "rolling" patch updates can be applied across your nodes without taking the cluster down. (We described the process of applying rolling patch updates in Chapter 3.) Further, you can roll back a patch (e.g., uninstall it) on an instance if you observe unusual behavior and want to remove the patch.

CHAPTER 6

Oracle Security, Auditing, and Compliance

The primary purpose of Oracle Database software is to manage the valuable data that lies at the core of virtually every operation in your organization. Part of the value of that data is that the data is *yours*—the data that can be used to give your company unique advantages. For this reason, you need to protect your data from others who should not have access to it. This protection is the subject of this chapter. Here we focus on three different aspects of the overall task of protecting your data:

Security
: Covers the tools and features that you use to allow access only to those people authorized to view or use specific data.

Auditing
: Allows you to discover who did what with your data. Auditing is the process of creating a history of access that can be used to understand database operations as well as spot access violations and attempts.

Compliance
: This is the ability to prove that your data is secure and reliable—a proof that is now legally required in many cases. Although compliance may strike many technical folks as overkill, the simple fact is that a lack of compliance alone may result in significant penalties to your company. Compliance is thus a topic of great interest to management.

Security

One of the most important aspects of managing the Oracle Database effectively in a multiuser environment is the creation of a security scheme to control access to and modification of the database. In an Oracle Database, you grant security clearance to individual users or database roles, as we describe in the following sections.

Security management is typically performed at three different levels:

- Database level
- Operating system level
- Network level

At the operating system level, DBAs should have the ability to create and delete files related to the database, whereas typical database users do not need these privileges. Oracle includes operating system-specific security information as part of its standard documentation set. In many large organizations, DBAs or database security administrators work closely with computer system administrators to coordinate security specifications and practices.

Database security specifications control user database access and place limits on user capabilities through the use of username/password pairs. Such specifications may limit the allocation of resources (disk and CPU) to users and mandate the auditing of users. Database security at the database level also provides control of the access to and use of specific schema objects in the database. We believe that implementing data security in the database is a best practice, rather than using security controls in applications or other layers of the technology stack. However, as you will see, there are features in the Oracle Database that can work with concepts like application users.

Usernames, Privileges, Groups, and Roles

The DBA or database security administrator creates *usernames* that can be used to connect to the database. Two user accounts are automatically created as part of the installation process and are assigned the DBA role: SYS and SYSTEM. (The DBA role is described in a later section.)

Each database username has a password associated with it that prevents unauthorized access. A new or changed password should:

- Contain at least eight characters
- Contain at least one number and one letter
- Not be the username reversed
- Differ from the username or user name with 1 through 100 appended
- Not match any word on an internal list of simple words
- Differ from the previous password (if there is one) by at least three characters
- Since 11*g*, passwords can require mixed cases for characters

Oracle can check for these characteristics each time a password is created or modified as part of enforced security policies. You can also set your own specifications for password complexity for your Oracle Database.

Once a user has successfully logged into the database, that user's access is restricted based on *privileges*, which are the rights to execute certain SQL commands. Some privileges may be granted system-wide (such as the ability to delete rows anywhere in the database), while others may apply only to a specific schema object in the database (such as the ability to delete rows in a specific table).

Roles are named groups of privileges and may be created, altered, or dropped. In most implementations, the DBA or security administrator creates usernames for users and assigns roles to specific users, thereby granting them a set of privileges. This is most commonly done today through the Oracle Enterprise Manager (EM) console, described in Chapter 5. For example, you might grant a role to provide access to a specific set of applications, such as "Human Resources," or you might define multiple roles so that users assigned a certain role can update hourly pay in the Human Resources applications, while users assigned other roles cannot.

Every database has a pseudorole named PUBLIC that includes every user. All users can use privileges granted to PUBLIC. For example, if database links are created using the keyword PUBLIC, they will be visible to all users who have privileges to the underlying objects for those links and synonyms. As database vulnerability is an increasing concern, you may want to consider limited privileges for the PUBLIC role.

Identity Management

As the number of database users and complexity of your database structures rises, managing user identities and their privileges can become increasingly complex. Oracle Identity Management can provide a solution by storing user information and their authorization in an LDAP directory such as the Oracle Internet Directory (OID). For example, you might use OID to authorize SYSDBA and SYSOPER connections. This centralization is even more useful when you use a centralized directory to manage security across multiple databases in your environment.

Security Privileges

Four basic types of database operations can be limited by security privileges in an Oracle Database:

- SELECT to perform queries
- INSERT to put rows into tables or views
- UPDATE to update rows in tables or views

- DELETE to remove rows from tables, table partitions, or views

In addition to these data-specific privileges, several other privileges apply to the objects within a database schema, such as:

- CREATE to create a table in a schema
- DROP to remove a table in a schema
- ALTER to alter tables or views

All of these privileges can be handled with two simple SQL commands. The `GRANT` command gives a particular privilege to a user or role, while the `REVOKE` command takes away a specific privilege. You can use `GRANT` and `REVOKE` to modify the privileges for an individual or a role. You can also grant the ability to re-grant privileges to others. You can use either of these commands with the keyword PUBLIC to issue or revoke a privilege for all database users.

Another security privilege, EXECUTE, allows users to run a PL/SQL procedure or function. By default, the PL/SQL routine runs with the security privileges of the user who compiled the routine. Alternately, you can specify that a PL/SQL routine run with what is termed *invoker's rights*, which means that the routine is run with the security privileges of the user who is invoking the routine.

Special Roles: DBA, SYSDBA, and SYSOPER

Your Oracle Database comes with three special roles, which have been defined for a while, and more roles added in the last releases, such as *sysasm* in Oracle Database 11*g* and *sysbackup* in Oracle Database 12*c*. The DBA role is one of the most important default roles in Oracle. The DBA role includes most system privileges. By default, it is granted to the users SYS and SYSTEM, both created at database creation time. Base tables and data dictionary views are stored in the SYS schema. SYSTEM schema tables are used for administrative information and by various Oracle tools and options. A number of other administrative users also exist, as consistent with the specific Oracle features deployed.

The DBA role does not include basic database administrative tasks included in the SYSDBA or SYSOPER system privileges. Therefore, SYSDBA or SYSOPER should be specifically granted to administrators. They will "CONNECT AS" either SYSDBA or SYSOPER to the database and will have access to a database even when it is not open. SYSDBA privileges can be granted to users by SYS or by other administrators with SYSDBA privileges. When granted, the SYSDBA privileges allow a user to perform the following database actions from the command line of SQL*Plus or by logging into Oracle Enterprise Manager's point-and-click interface:

`STARTUP`
: Start up a database instance.

`SHUTDOWN`
: Shut down a database instance.

`ALTER DATABASE OPEN`
: Open a mounted but closed database.

`ALTER DATABASE MOUNT`
: Mount a database using a previously started instance.

`ALTER DATABASE BACKUP CONTROLFILE`
: Start a backup of the control file. However, backups are more frequently done through RMAN today, as described in the section "Backup and Recovery" on page 151 in Chapter 5.

`ALTER DATABASE ARCHIVELOG`
: Specify that the contents of a redo logfile group must be archived before the redo logfile group can be reused.

`ALTER DATABASE RECOVER`
: Apply logs individually or start automatic application of the redo logs.

`CREATE DATABASE`
: Create and name a database, specify datafiles and their sizes, specify logfiles and their sizes, and set parameter limits.

`DROP DATABASE`
: Delete a database and all of the files included in the control file.

`CREATE SPFILE`
: Create a server parameter file from a text initialization (*INIT.ORA*) file.

`RESTRICTED SESSION` *privilege*
: Allow connections to databases started in Restricted mode. Restricted mode is designed for activities such as troubleshooting and some types of maintenance, similar to what SYS can do, but limited to a different set of users.

Administrators connected as SYSOPER can perform a more limited set of commands: `STARTUP` and `SHUTDOWN`, `CREATE SPFILE`, `ALTER DATABASE OPEN` or `MOUNT` or `BACKUP`, `ALTER DATABASE ARCHIVELOG`, `ALTER DATABASE RECOVER`, and the `RESTRICTED SESSION` privilege.

Database administrators are authenticated using either operating system authentication or a password file. The CONNECT INTERNAL syntax supported in earlier releases of Oracle is no longer available. When operating system authentication is used, administrative users must be named in the OSDBA or OSOPER defined groups. For password

file authentication, the file is created with the ORAPWD utility. Users are added by SYS or by those having SYSDBA privileges.

With each release of Oracle, fewer default users and passwords are automatically created during database installation and creation. Regardless, it is generally recommended practice to reset all default passwords that are documented in Oracle.

These special roles are very powerful, granting broad powers to users. Some organizations have granted these system roles to users who may not need all that power. Oracle Database 12*c* includes the ability to analyze privileges to identify users with broad privileges who do not need them.

Policies

A *policy* is a way to extend your security framework. You can specify additional requirements in a policy that are checked whenever a user attempts to activate a role. Policies are written in PL/SQL and can be used, for example, to limit access to a particular IP address or to particular hours of the day.

Since the release of Oracle Database 10*g*, Oracle Enterprise Manager has featured a visual interface to a policy framework in the EM repository that aids management of database security. Security policies or rules are built and stored in a policy library. Violations of rules are reported as critical, warning, or informational through the EM interface. Out of the box, security violations are checked on a daily basis. Policies may be adjusted according to business demands, and violations can be overridden when they are reported.

Restricting Data-Specific Access

There are situations in which a user will have access to a table, but not all of the data in the table should be viewed. For example, you might have competing suppliers looking at the same tables. You may want them to be able to see the products they supply and the total of all products from suppliers, but not detailed information about their competitors. There are a number of ways to do this, as we'll describe in the following sections, using other examples from Human Resources (HR).

View-based security

You can think of *views* as virtual tables defined by queries that extract or derive data from physical *base tables*. You can use views to present only the rows or columns that a certain group of users should be able to access.

For example, in an HR application, users from the HR department may have full access to the employee base table, which contains basic information such as employee names, work addresses, and work phone numbers, as well as more restricted information such as Social Security numbers, home addresses, and home telephone numbers. For other users in the company, you'll want to hide more personal information by providing a view that shows only the basic information.

Creating a virtual private database or leveraging the Label Security Option, described in subsequent sections of this chapter, provide a more secure means of restricting access to certain data.

Fine-grained access control

Implementing security is a critical but time-consuming process, especially if you want to base security on an attribute with a wide range of values. A good example of this type of situation in the HR scenario previously described would be the need to limit the data an HR representative can see to only the rows relating to employees that he supports. Here you're faced with a situation in which you might have to define a view for every HR representative, which might mean many, many different views, views that would have to change every time an HR representative left or joined the company. And if you want to grant write access for a representative's own employees and read access for other employees, the situation gets even more complex. The smaller the scope, or *grain*, of the access control you desire, the more work is involved in creating and maintaining the security privileges.

Oracle offers a type of security that you can use to grant this type of *fine-grained access control* (FGAC). *Security policies* implemented as PL/SQL functions can be associated with tables or views enabling creation of a virtual private database (VPD). A security policy returns a condition that's dynamically associated with a particular SQL statement, which transparently limits the data that's returned. In the HR example, suppose that each representative supports employees with a last name in a particular alphabetic range, such as A through G.

The security policy would return part of a WHERE clause, based on a particular representative's responsibilities, that limits the rows returned. You can keep the range for each representative in a separate table that is dynamically queried as part of the security policy function. This simplifies management of allowable access if roles and responsibilities change frequently.

You can associate a security policy with a particular view or table by using the built-in PL/SQL package DBMS_RLS, which also allows you to refresh, enable, or disable a security policy.

Oracle Database 10*g* and newer database releases feature a VPD that is even more fine-grained, enabling enforced rewrites when a query references a specific column. Performance of queries in VPD implementations was also improved since Oracle Database

10*g* through the support of parallel query. Fine-grained security can also be based on the type of SQL statement issued. The security policy previously described could be used to limit UPDATE, INSERT, and DELETE operations to one set of data, but allow SELECT operations on a different group of data. For a good description of FGAC through PL/SQL, please refer to: Oracle PL/SQL Programming by Steven Feuerstein and Bill Pribyl and Oracle PL/SQL for DBAs by Arup Nanda and Steven Feuerstein (O'Reilly).

Label Security Option

The Oracle Label Security Option eliminates the need to write VPD PL/SQL programs to enforce row-level label security where sensitivity labels are desired. The collections of labels, label authorizations, and security enforcement options can be applied to entire schemas or to specific tables.

Sensitivity labels are defined based on a user's need to see and/or update data. They consist of a level denoting the data sensitivity, a category or compartment that further segregates the data, and a group used to record ownership (which may be hierarchical in nature) and access.

Standard group definitions given to users provide them access to data containing those group labels. Inverse groups in the data can be used to define what labels a user must have in his profile in order to access it.

Policies are created and applied, sensitivity labels are defined, and user labels are set and authorized through a policy manager tool accessible through EM. You can also add SQL predicates and label functions and manage trusted program units, Oracle VPD fine-grained access control policies, and VPD application contexts. Label Security policy management is possible in Oracle Database 10*g* and later versions when the Oracle Internet Directory is also used.

Security and Application Roles and Privileges

Applications can involve data and logic in many different schemas with many different privileges. To simplify the issues raised by this complexity, roles are frequently used in applications. Application roles have all the privileges necessary to run the applications, and users of the applications are granted the roles necessary to execute them.

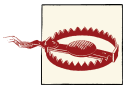
Application roles may contain privileges that should be granted to users only while they're running the application. Application developers can place a SET ROLE command at the beginning of an application to enable the appropriate role and disable others only while the application is running. Similarly, you can invoke a DBMS_SESSION.SET_ROLE procedure from PL/SQL.

Another way application security is sometimes accomplished is by encapsulating privileges in stored procedures. Instead of granting direct access to the various tables for an

application, you can create stored procedures that provide access to the tables and grant access to the stored procedures instead of the tables. For example, instead of granting INSERT privileges for the EMPLOYEE table, you might create and grant access to a stored procedure called HIRE_EMPLOYEE that accepts as parameters all the data for a new employee.

When you run a stored procedure normally, the procedure has the access rights that were granted to the owner of the procedure; that owner is the schema in which the procedure resides. If a particular schema has access to a particular database object, all stored procedures that reside in that schema have the same rights as the schema. When any user calls one of those stored procedures, that user has the same access rights to the underlying data objects that the procedure does.

For example, suppose there is a schema called HR_REP. This schema has write access to the EMP table. Any stored procedure in the HR_REP schema also has write access to the EMP table. Consequently, if you grant a user access to a stored procedure in the HR_REP schema, that user will also have write access to the EMP table regardless of her personal level of security privilege. However, she will have access only through the stored procedures in the schema.



One small but vitally important caveat applies to access through stored procedures: the security privilege must be *directly* granted to the schema, not granted by means of a role.

If you attach the keyword AUTHID CURRENT_USER to a stored procedure when it is compiled, security restrictions will be enforced based on the username of the user invoking the procedure, rather than the schema that owns the stored procedure (the definer of the procedure). If a user has access to a particular database object with a particular privilege, that user will have the same access through stored procedures compiled with the AUTHID CURRENT_USER.

Oracle Database 12*c* adds a whole new concept called Real Application Security, which is similar to a VPD but more flexible. With Real Application Security, you can define users outside the context of a database user and assign security privileges based on that user identity. This separation allows for more flexibility, as you do not need to have a database user defined for every application user, as well as the added flexibility of policy-based security definition. Real Application Security maps to a security architecture where data security is enforced in the database, and users are defined and authenticated in the context of the application.

Distributed Database and Multitier Security

All the security features available for standard Oracle Databases are also available for the distributed database environment, which is covered in Chapter 13. However, the

distributed database environment introduces additional security considerations. For example, user accounts needed to support server connections must exist in all of the distributed databases forming the system. As database links (which define connections between distributed database instances) are created, you will need to allow the user accounts and roles needed at each site.

Distributed security management

For large implementations, you may want to configure global authentication across these distributed databases for users and roles. Global authentication allows you to maintain a single authentication list for multiple distributed databases. Where this type of external authentication is required, Oracle's Advanced Security Option, discussed in the next section, provides a solution.

Enterprise Manager is commonly used to configure valid application users to Oracle's LDAP-compliant OID server. A user who accesses an application for which he is not authenticated is redirected to a login server. There, he is prompted for a username and password that are checked against the OID server. A cookie is returned and the user is redirected from the login server to the application.

Oracle Identity Management, described earlier in this chapter, can be used to manage security across multiple platforms and security systems.

Multitier security

In typical three-tier implementations, the Oracle WebLogic Server runs some of the application logic, serves as an interface between the clients and database servers, and provides much of the Oracle Identity Management (OIM) infrastructure. The Oracle Internet Directory provides directory services running as applications on an Oracle Database. The directory synchronization service, provisioning integrated service, and delegated administrative service are part of OID. Security in middle-tier applications is controlled by applications' privileges and the preservation of client identities through all three tiers.

Deploying multiple tiers, common with large applications or web-based applications, can also call for proxy authentication. The application connects to code in the middle tier, which accesses the database through a proxy, frequently through shared connections. Some databases associate security with a session, which means that sessions must be reestablished when the user identity changes. This limitation makes the multitier approach harder.

Oracle separates authentication from sessions, so the use of a proxy in the middle tier is feasible. A single session can support different users with different identities. Prior to Oracle 10*g* Release 2, the only way to take advantage of this capability was by using the OCI interface, which was code-intensive. Since Oracle Database 10*g* Release 2, this limitation was lifted, so standard SQL and SQL tools, such as SQL*Plus, could use proxy authentication.

Advanced Security Option

The Oracle Advanced Security Option (ASO) can be used in distributed environments linked via Oracle Net in which there are concerns regarding secure access and transmission of data. This option specifically provides data encryption during transmission to protect data from unauthorized viewing over Oracle Net, as well as Net/SSL, IIOP/SSL, and between thin JDBC clients and the database. Encryption algorithms supported include RC4_40, RC4_56, RC4_128, RC4_256, DES, DES_40, 3DES112, 3DES168, AES128, AES192, and AES256. Communications packets are protected against data modification, transaction replay, and removal through use of MD5 and SHA-1 algorithms. Network encryption is slated to be moved into Oracle Database 12*c* Enterprise Edition.

Transparent Data Encryption (described in the next section) is included as part of the Advanced Security Option beginning with Oracle Database 10*g* Release 2. Transparent Data Encryption provides an easy way to encrypt data in the database, and the network data encryption option of ASO protects the data during transmission to the client.

ASO also provides support for a variety of identity authentication methods to ensure that user identities are accurately known. Third-party authentication services supported include Kerberos, RADIUS, and DCE. RADIUS enables support of third-party authentication devices, including smart cards and token cards. Public Key Infrastructure (PKI) authentication, popular for securing Internet-based e-commerce applications, uses X.509 v3 digital certificates and can leverage Entrust Profiles stored in Oracle Wallets. Oracle Database 10*g* added authentication capabilities for users who have Kerberos credentials, and enables Kerberos-based authentication across database links. Strong authentication methods are also being moved into Oracle Database 12*c*.

In a typical scenario, the Oracle Enterprise Security Manager configures valid application users to the LDAP-compliant OID server. An X.509 certificate authority creates private key pairs and publishes them in Oracle Wallets (through Oracle Wallet Manager) to the LDAP directory. A user who wants to log in to a database server will need a certificate and a private key, which can be retrieved from that user's password-protected wallet, which resides in the LDAP directory. When the user's key on the client device is sent to the database server, it is matched with the paired key retrieved by the server via SSL from the LDAP directory and the user is authenticated to use the database.

Oracle Database 12*c* includes a number of enhancements for the Advanced Security Option, including the ability to manage keys more easily. In this release, the master key for encryption is associated with a pluggable database, so you could potentially have multiple master keys in a single container database.

Encryption

The previous sections of this chapter all deal with the need to protect access to data in the Oracle Database. There may be times when you want to take the extra step of protecting the actual data values from unauthorized viewing by encrypting the data.

Oracle has provided data encryption for several releases, but Oracle Database 10*g* Release 2 first introduced a significant new feature called Transparent Data Encryption. Prior to the introduction of this feature, encrypted data stored in the Oracle Database had to be decrypted by an application before it could be used. This scenario caused a number of limitations, such as the need to explicitly decrypt data in all applications that used the encrypted data and the possibility that some SQL options, such as sorting, would not work as expected. If you wanted to start encrypting a particular piece of data, you would have to change all data access routines in every application that used the data. This limitation alone made it difficult to consider adding encryption to existing data.

With Transparent Data Encryption, the database does the work of encrypting and decrypting data automatically. Data sent to the database is encrypted by Oracle, and data requested from the database is decrypted. No additional code is required in an application, which means that you can encrypt existing data without changing any of your SQL access statements.

Since Oracle Database 11*g*, you can encrypt entire tablespaces (described in Chapter 4) with Transparent Data Encryption, and this feature should reduce management overhead for this feature.

Data Redaction

Oracle Database 12*c* adds another option for obscuring data. *Data redaction* is the ability to return data with different values than the actual value stored in the database, such as returning "XXXX-XXXX-XXXX-4239" for a credit card number instead of the stored value of "1234-5678-9012-4239." Data redaction in Oracle Database 12*c* allows you to implement redaction policies, which can specify users and conditions to redact data. You can completely redact values, partially redact values, redact values based on a regular expression to maintain formatting, or perform random redaction.

This redaction is frequently transparent to the use of the data by applications. The data redaction also maintains the proper data format for the affected data.

Data redaction is managed for Oracle Database 12*c* through Enterprise Manager, SQL Developer, or through a command line interface.

Secure Backup

The security features described in previous sections give you the tools you need to keep the data in your Oracle Database secure. But what about when the data leaves your

Oracle database—for example, when you perform the necessary maintenance step of backing up the data?

Lost backup tapes are always a possibility, and backup tapes can be stolen. Secure Backup, first released between Oracle Database 10*g* Release 2 and Oracle Database 11*g*, automatically encrypts your backup data. The data can be decrypted only by the source database, so even if a backup tape is lost or stolen, the recipient will not be able to see your data.

Auditing

The Oracle Database gives you the ability to restrict unauthorized access to your valuable data. However, your security is only as good as your implementation, and people do make mistakes. In addition, you may want to understand what type of activities—legitimate or not—are taking place with your data. The ability to audit database activity can address both of these issues.

Oracle's audit capabilities let you track actions at the statement level, privilege level, or schema object level for the entire database or particular users. Auditing can also gather data about database activities for planning and tuning purposes. Auditing of connections with administrative privileges to an instance and audit records recording database startup and shutdown occur by default.

You can also audit sessions at the user level, which captures some basic but extremely useful statistics such as the number of logical I/Os, the number of physical I/Os, and the total time logged on. As noted in the previous chapter, gathering performance statistics is low in terms of collection overhead, and Oracle Database 10*g* and later releases automatically gather statistics in populating the Automatic Workload Repository (AWR).

Audit records always contain the following information:

- Username
- Session identifier
- Terminal identifier
- Name of schema object accessed
- Operation performed or attempted
- Completion code of the operation
- Date and timestamp

The records may be stored in a data dictionary table (AUD$ in the SYS schema), which is also called the database audit trail, or in an operating system audit trail.

Oracle9*i* added fine-grained auditing, which enabled selective audits of SELECT statements with bind variables based on access of specified columns. Oracle Database 10*g* added extended SQL support for fine-grained auditing. You can now perform granular auditing of queries, and UPDATE, INSERT, and DELETE operations through SQL.

With Oracle Database 12*c*, auditing is policy-based, which gives you even more flexibility and ease of use. You can choose when to save audit records based on conditions, as well as allowing user exceptions for audit record collection.

Since Oracle Database 11*g*, auditing is turned on by default, and the AUDIT_TRAIL initialization parameter is set to DB. Privileges audited by default include:

- ALTER ANY PROCEDURE
- ALTER ANY TABLE
- ALTER DATABASE
- ALTER PROFILE
- ALTER SYSTEM, ALTER USER
- AUDIT SYSTEM
- CREATE ANY JOB, CREATE ANY LIBRARY, CREATE ANY PROCEDURE, CREATE ANY TABLE, CREATE EXTERNAL JOB, CREATE PUBLIC DB LINK, CREATE SESSION, CREATE USER
- DROP ANY PROCEDURE, DROP ANY TABLE, DROP PROFILE, DROP USER
- EXEMPT ACCESS POLICY
- GRANT ANY OBJECT PRIVILEGE, GRANT ANY PRIVILEGE, and GRANT ANY ROLE

Oracle Database 12*c* gives you the option of requiring immediate audit record writes or allowing periodic writes of audit records every few seconds, which can lower the impact of auditing on resource consumption. Each pluggable database has its own audit trails and set of audit policies, which are automatically moved if you move the pluggable database.

Compliance

The slogan "trust, but verify" could apply to the functions of security and auditing. Compliance extends that slogan to "trust, verify, and prove it" and describes the tools necessary to provide proof that your data has been used properly.

Compliance is based on the security and audit features described in previous sections. For the most part, compliance is the result of a new element introduced into the corporate landscape—government requirements. In the United States and elsewhere,

compliance is being increasingly required by government regulation, so the ability of the Oracle Database to make compliance easy is becoming correspondingly important. Compliance is crucial for many organizations, and the people responsible for guaranteeing compliance are not necessarily in the IT department. Consequently, the implementation of security and audit schemes has had to be simplified and coordinated to address compliance needs.

Oracle has two options specifically designed to address compliance challenges—Oracle Data Vault and Oracle Audit Vault; these are described in the following sections. The related Flashback Data Archive capability, also mentioned below, is described in greater detail in Chapter 3. In addition, privilege analysis, mentioned above, can be used to demonstrate the extent of privilege grants to auditors.

Oracle Database Vault Option

The Oracle Database Vault Option was introduced in 2006 and restricts DBAs and other highly privileged users from accessing application data to which they should not have access. It can also be set up so that applications' DBAs are not allowed to manipulate the database or access other applications. A security administrator can use the Oracle Database Vault Option to describe the security scheme that the organization wants to implement, and this option automatically implements the schemes using the features described earlier in this chapter.

Key parameters defined in the Oracle Database Vault Option are called *factors*. A factor is essentially a descriptive dimension that will affect security across the entire database. Factors include things such as specific application programs, locations, or times of day. This option comes with more than 40 factors defined, and users can create their own factors.

Factors are used to define access and audit particular security dimensions. You can create rules that limit types of access to a particular factor and rule sets that combine multiple factor rules together. Once you have defined rule sets, you can create application roles based on these sets, as well as command rules that control whether database commands can be executed, based on the outcome of rule evaluation. For example, you could prevent anyone from dropping a particular table unless the command came from a particular location defined by a factor, or specify that new users can be defined only by the combined actions of two administrators.

Rules can also be used to define database *realms*, which consist of a subset of the schemas and roles that an administrator can administer. This ability is essential if an organization uses its Oracle Database to service multiple communities. You can define a realm and give an administrator privileges on that realm without compromising data in other schemas. The overall effect of realms is to allow secure delegation of administrative responsibilities.

Mandatory realms were introduced with Oracle Database 12*c*. Prior to this release, realms only operated on general system privileges used for specific tables. If you eliminated the use of SELECT on your PAYROLL table, that keyword would not be allowed for users who gained access to the table through a privilege like SYSDBA. With a mandatory realm, you can also block access for users who have been given that access through direct `GRANT` operations.

The installation and management of Database Vault has also been made significantly easier with Oracle Database 12*c*, allowing installation with just two commands.

All of the rule enforcement is audited as part of the Oracle Database Vault Option, which provides the type of documentation required for complete compliance. Figure 6-1 illustrates the various components of the Oracle Database Vault Option solution.

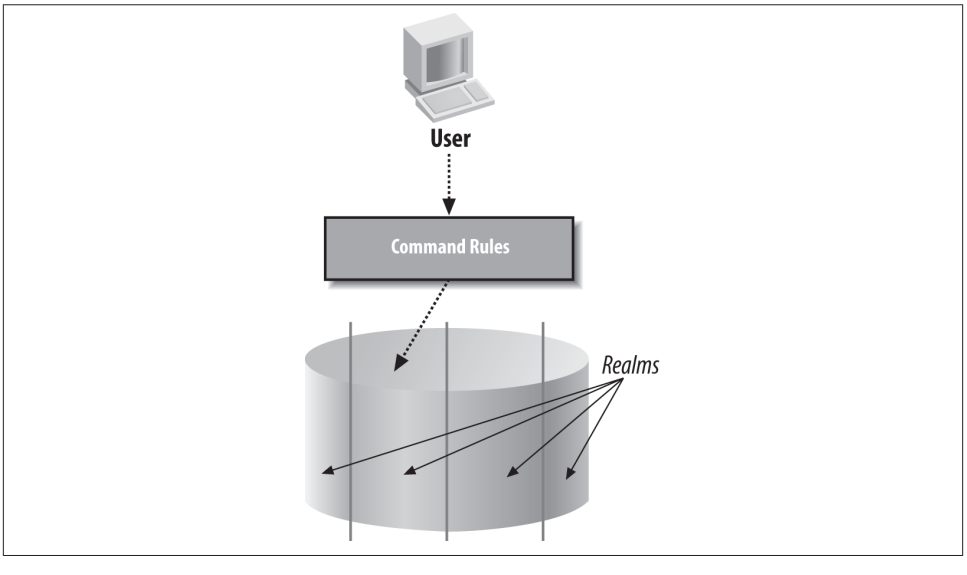


Figure 6-1. Oracle Database Vault Option components

Oracle Audit Vault Server

The Oracle Audit Vault Server was introduced in 2007 and collects data from audit files in Oracle and in the underlying operating system. It consolidates this data in a secure repository and provides out-of-the-box compliance reporting. Among the reports provided are privileged user accesses, account management, data access, and failed login attempts. Stored in an Oracle data warehouse schema, the data is easily accessible by business intelligence tools such as Oracle's BI Publisher.

Because the Oracle Audit Vault Server monitors all incoming audit data, it can generate alerts based on IT policies. For example, policies can be defined to trigger alerts for

privileged users' changes and sensitive data access. Oracle Databases dating back to Oracle 9*i* Release 2 can be monitored. A software development kit (SDK) is available for building custom audit collectors.

Oracle acquired a product now known as Database Firewall in 2010. The product performs traditional network firewall operations, but with a focus on limiting access to database capabilities, such as SQL injection. In Oracle Database 12*c*, the Database Firewall product has been added to Audit Vault.

Flashback Data Archive

Flashback technology was introduced in Chapter 3, because this capability is based on rollback segments. Although Flashback was initially introduced with Oracle9*i*, Oracle Database 11*g* first enabled a particular use of Flashback that can help address compliance issues.

Flashback Data Archive gives you the ability to see all of the changes that occur to a record throughout its lifetime. This type of history tracking can provide the key information required to demonstrate compliance, as well as to track the source of errors in compliance or usage.

Transparent Sensitive Data Protection

Tied to the Oracle Database 12*c* release, Enterprise Manager includes a new capability called Sensitive Data Discovery. This procedure helps to discover the existence of potentially sensitive data by examining a number of sources, including the data dictionary and metadata for applications, and uses information such as data relationships to identify sensitive data that might need attention, such as encryption, redaction, or encrypting. All data security features in Oracle Database 12*c* can use this information in a feature called Transparent Sensitive Data Protection to create policies to protect the sensitive data. The policy is uniformly implemented over all occurrences of a particular type of sensitive data, and the policies can be changed to keep in step with changing audit and protection requirements.

CHAPTER 7
Oracle Performance

This book illustrates the wide range of features that the Oracle Database has. As you gain experience with Oracle, you'll reap more of the benefits it has to offer. One area on which you will eventually focus is performance tuning, since you will inevitably be forced to wring additional performance from your Oracle Database in the face of increasing demands. This chapter covers the basics you'll need to understand as you address performance.

Oracle Database performance tuning has been extensively documented in the Oracle community. There are numerous books that provide detailed and excellent information. This book is focused more on the concepts of the Oracle Database, so we won't delve too deeply into specific tuning recommendations. Instead, we'll touch on the importance of tuning and discuss some basic notions of how Oracle uses resources. Here, we're simply laying a foundation for understanding Oracle performance. This understanding will help you implement the tuning procedures most suited for your own particular implementation. Where appropriate, we'll provide some basic guidance on how the latest Oracle Database features and Enterprise Manager help you manage performance.

Performance is one of the trickiest aspects in the operation of your database since so many factors can be involved. There is the Oracle Database, to be sure. But there are also platform deployment strategies to consider. Today, the infrastructure for your applications likely resides on multiple platforms, including database servers and applications servers. There is network and interconnect bandwidth to consider and varying complexity in use among your users.

One of the curious aspects of performance is that "good performance" is defined by its absence rather than by its presence. You can recognize bad performance easily, but good performance is usually defined as simply the absence of bad performance. Performance is simultaneously a very simple topic—any novice user can implicitly understand it—

and an extremely complex topic that can strain the ingenuity of the most proficient database professional.

Certainly, Oracle provides more and better automated tuning options in current releases than it did when we wrote earlier editions of this book. These can help you better manage contention, identify high-load SQL, tune initialization parameters, and identify problems caused by badly designed applications. However, getting optimal performance also relies on proper configuration of your server and storage platform. There is no substitute for getting your hardware platform properly configured with appropriate CPUs and cores, memory, and especially storage and I/O (throughput).

We'll begin this chapter by describing how Oracle uses system resources and how to efficiently use these resources to gain performance. Then we'll cover some of the basic features and tools used for performance tuning the Oracle Database.

Oracle and Resource Usage

Chapter 12 covers some of the basics of good server and storage configurations and explains how Oracle-engineered systems provide a possible means of assuring balanced configurations. You are more likely to run into performance issues and face difficult tuning challenges if inadequate server and storage resources are available to the Oracle Database. Problems such as those caused by inadequate I/O, CPUs and CPU cores, or memory are still common today where unbalanced configurations are deployed. Where data is being accessed in a federated fashion across multiple servers, network bandwidth can also play a part in performance challenges. However, most organizations avoid this problem by minimizing use of such federated strategies.

Network bandwidth can become a concern when using your Oracle Database to retrieve very large data sets over the network. Although you can't typically surmount this type of problem simply by improving the performance of your Oracle Database, you can monitor network and application server bottlenecks with Oracle Enterprise Manager.

If your Oracle Database is not properly designed and configured, you could also have more fundamental problems. We will next focus on how Oracle uses the three key machine resources: CPU, memory, and disk I/O. The slowest access is to disk and, as a result, the most common database performance issues are I/O-related. The majority of this section therefore focuses on performance as it relates to physical disk I/O.

A database server may experience bottlenecks caused by contention for multiple resources at the same time. In fact, where platforms are managed such that one resource will try to compensate for the lack of another resource, there is sometimes a deficit in

the compensating resource as well. For example, if you run out of physical memory, the operating system might swap areas of memory out to the disk and can cause I/O bottlenecks.

Oracle and Disk I/O Resources

From the perspective of machine resources, an input/output operation, or I/O, can be defined as the operating system of the computer reading or writing some bytes from or to the underlying disk subsystem of the database server. I/Os can be small, such as 4 KB of data, or large, such as 64 KB or 128 KB of data. The lower and upper limits on the size of an I/O operation vary according to the operating system. The Oracle Database also has a block size that you can define, called the *database block size*. The default size for Oracle is typically 4 KB or 8 KB depending on operating system.

An Oracle Database can issue I/O requests in two basic ways:

Single database block I/Os
: For example, one 8 KB datablock I/O request at a time. This type of request reads or writes a specific block. After looking up a row in an index, Oracle uses a single block I/O to retrieve the desired database block.

Multiblock I/Os
: For example, 32 database blocks, each consisting of 8 KB, for a total I/O size of 256 KB. Multiblock I/O is used for large-scale operations. The number of blocks in one multiblock I/O is determined by the initialization parameter DB_FILE_MULTIBLOCK_READ_COUNT. Setting this value too high will favor full table scans.

The Oracle Database can read larger amounts of data with multiblock I/Os, so there are times when a full table scan might actually retrieve data faster than an index-based retrieval (e.g., if the selectivity of the index is low). Oracle can perform multiblock operations faster than the corresponding collection of single-block operations.

I/O Planning Principles for an Oracle Database

When you're planning the disk layout and subsequent placement of the various files that make up your database, you need to consider why Oracle performs I/O and the potential performance impacts.

The amount of I/O is affected by the following in Oracle:

- Redo logs
- Data contained in tables
- Indexes on the tables
- The data dictionary, which goes in the SYSTEM tablespace

- Sort activity, which goes in the TEMP tablespace of the user performing the sort when outside the boundary of the sort area size
- Rollback information, which is spread across the datafiles of the tablespace containing the database's rollback segments
- Archived redo logs, which go to the archived log destination (assuming the database is in ARCHIVELOG mode)

Striping of data across disk is used to spread I/O evenly across multiple spindles and speed performance. Striping data can assure I/O operations occur across multiple spindles, reducing contention on individual drives, and speeding database performance. For example, suppose you were to place a datafile containing an index on a single drive. If multiple processes use the index simultaneously, they will all issue I/O requests to the one disk drive, resulting in contention for the use of that drive.

Instead, suppose you placed the same datafile on a "disk" that was actually an array of five physical disks. Each physical disk in the array can perform I/O operations independently on different data blocks of the index, automatically increasing the amount of I/O Oracle can perform without causing contention.

Since Oracle Database 10*g*, Oracle's Automatic Storage Management (ASM) provides automatic striping and rebalancing of stripe sets. ASM also provides mirroring for high availability and is typically accessed for managing storage through Enterprise Manager.

ASM divides files into extents and spreads the extents evenly across each disk group. Pointers are used to track placement of each extent (instead of using a mathematical function such as a hashing algorithm to stripe the data). So when the disk group configuration changes, individual extents can be moved. In comparison to traditional algorithm-based striping techniques, the need to rerun that algorithm and reallocate all of the data is eliminated. Extent maps are updated when rebalancing the load after a change in disk configuration, opening a new database file, or extending a database file by enlarging a tablespace. By default, each extent is also mirrored, so management of redundancy is also simplified. Mirroring can be extended to triple mirroring or can be turned off. Although you still have to consider how many disk groups to use, implementation of these groups with striping and redundancy is automated with ASM.

Other simple principles for managing I/O have been used by DBAs to optimize Oracle's use of the database server's disk subsystem. Some of these include:

Use tablespaces to clearly segregate and target different types of I/O
: Separate table I/O from index I/O by placing these structures in different tablespaces. You can then place the datafiles for these tablespaces on various disks to provide better performance for concurrent access.

  Using tablespaces to segregate objects also simplifies tuning later on. Oracle implements I/O activity at the level of the datafile, or the physical object the operating

system sees as a file, and each file is a part of only one tablespace, as described in Chapter 4. Placing specific objects in specific tablespaces allows you to accurately measure and direct the I/O for those objects by tracking and moving the underlying datafiles as needed.

For example, consider a database with several large, busy tables. Placing multiple large tables in a single tablespace makes it difficult to determine which table is causing the I/O to the underlying datafiles. Segregating the objects allows you to directly monitor the I/O associated with each object. Your Oracle documentation details the other factors to consider in mapping objects to tablespaces.

Place redo logs and redo log mirrors on the two least-busy devices
: This placement maximizes throughput for transactional systems. Oracle writes to all copies of the redo logfile, and this I/O is not completed until all copies have been successfully written to. If you have two copies of the redo logfile, one on a slow device and the other on a fast device, your redo log I/O performance will be constrained by the slower device.

Distribute "system overhead" evenly over the available drives
: System overhead consists of I/O to the SYSTEM tablespace for the data dictionary, the TEMP tablespace for sorting, and the tablespaces that contain rollback segments for undo information. You should consider the system profile in spreading the system overhead over multiple drives. For example, if the application generates a lot of data changes versus data reads, the I/O to the rollback segments may increase due to higher writes for changes and higher reads for consistent read functionality.

Use a different device for archiving and redo logfiles
: To avoid archiving performance issues due to I/O contention, make sure that the archive log destination uses different devices from those used for the redo logs and redo log mirrors.

Volume managers

Oracle began providing its own volume manager software for Linux and Windows with Oracle9*i* Release 2. Since Oracle Database 10*g*, all Oracle Database releases for all supported operating systems include a cluster filesystem and volume manager in the database that is leveraged by ASM. When using ASM, an operating system volume manager will do little to improve database performance.

Older Oracle Database releases were often deployed on operating system-specific non-Oracle logical volume managers (LVM). An LVM acts as an interface between the operating system that requests I/O and the underlying physical disks. Volume-management software groups disks into arrays, which are then seen by the operating system as single "disks." The actual disks are usually individual devices attached to controllers or disks contained in a prepackaged array containing multiple disks and

controllers. This striping is handled by the volume management software and is completely transparent to Oracle. Figure 7-1 illustrates host-based volume management.

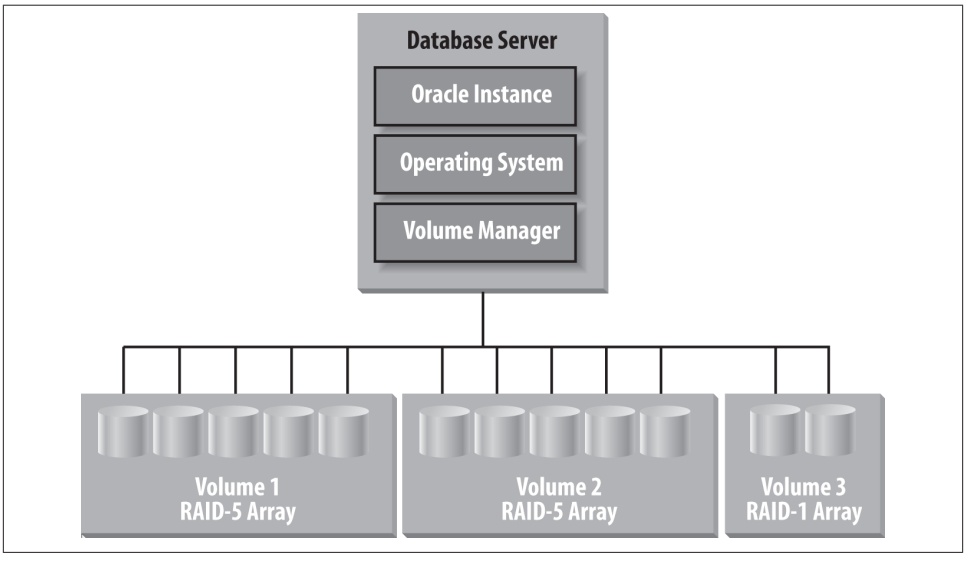


Figure 7-1. Host-based volume management

Storage subsystems

Storage subsystems, often referred to as disk farms, contain disks, controllers, CPUs, and (usually) memory used as an I/O cache. Examples of providers of such storage subsystems include Oracle, EMC, Network Appliance, Hewlett-Packard, IBM, and others. These subsystems offload the task of managing the disk arrays from the database server. The I/O subsystem is attached to the server using controllers. These dedicated storage devices are sometimes grouped into *storage area networks* (SANs) to denote their logical organization as a separate set of networked devices. The disk arrays are defined and managed within the dedicated I/O subsystem, and the resulting logical "disks" are seen by the operating system as physical disks.

This type of disk volume management is completely transparent to the database server and offers the following characteristics:

- The database server does not spend CPU resources managing the disk arrays.
- The I/O subsystem uses memory for an I/O cache, so the performance of Oracle I/O can improve.
- Write I/O is completed as soon as the data has been written to the subsystem's cache.
- The I/O subsystem will de-stage the data from cache to actual disk later.

- Read I/O can be satisfied from the cache. The subsystem can employ some type of algorithm to sense I/O patterns and preload the cache in anticipation of pending read activity.

Note that you must back up the cache in the subsystem with some type of battery so a power failure doesn't result in the loss of data that was written to the cache, but hasn't yet been written to the physical disk. Otherwise, data that Oracle assumes made it to disk may be lost, thereby potentially corrupting the database. Figure 7-2 illustrates a database server with a dedicated I/O subsystem.

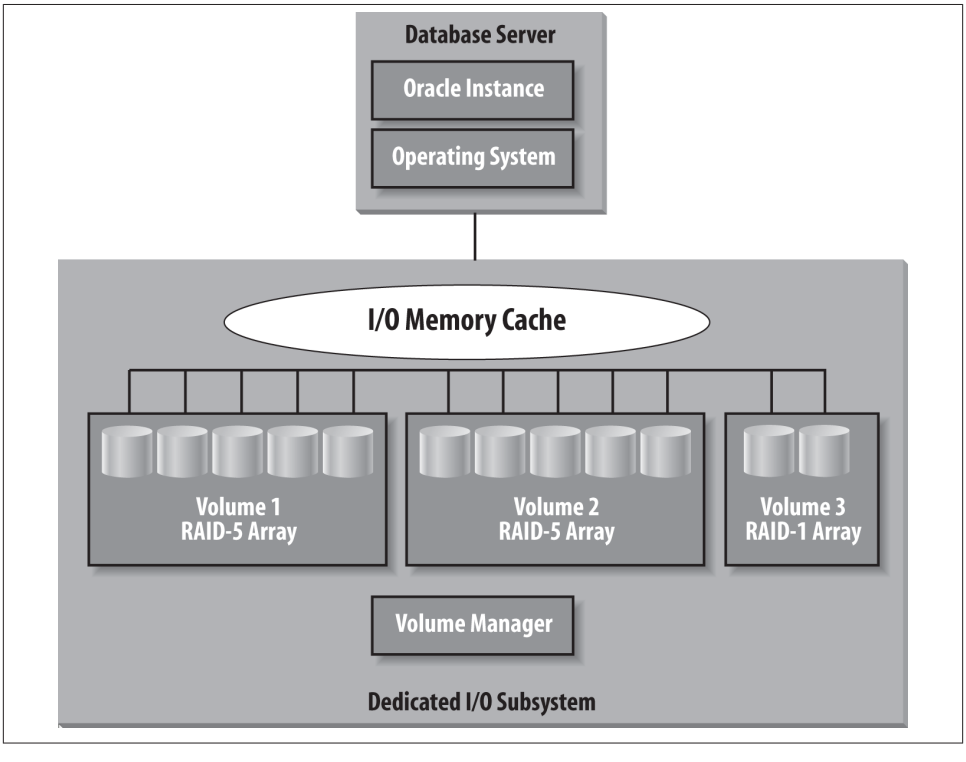


Figure 7-2. Dedicated I/O subsystems

Oracle's engineered systems and storage

Oracle's highest performing engineered systems for the Oracle Database, the Oracle Exadata Database Machine and the SuperCluster contain dedicated Exadata Storage Server cells. Why did Oracle use this approach?

For many years, getting I/O configurations right for storage attached to Oracle Database servers was problematic due to the number of vendors involved and the lack of focus on the potential negative performance impact of mismatched storage to workloads and

servers. Suffice it to say, defining a balanced server and storage combination where Database Server node performance is matched to Exadata Storage Server cells and linked via a high-speed interconnect (InfiniBand) greatly helps. In addition, Exadata's enabling of unique in-storage automated Oracle Database optimization techniques, storage indexes, and the Smart Flash Cache greatly speeds performance and simplifies challenges IT faces in designing optimally performing solutions. The use of balanced systems, with high bandwidth interconnects, and the "special sauce" of Exadata Storage Server software have greatly reduced the potential for I/O bottlenecks, which were previously the leading cause of reduced performance.

Where Oracle's engineered systems are deployed, other storage subsystems are still sometimes attached as part of an Information Lifecycle Management (ILM) strategy for archiving data online. Obviously, performance requirements when accessing archived data are much less demanding.

Oracle and Parallelism

The ability to parallelize operations is one of the most important features of the Very Large Database (VLDB). Database servers or nodes with multiple CPUs and CPU cores are the norm today for database servers. Oracle supports parallelism within single servers and nodes. Oracle supports further parallelism across multiple node configurations using Oracle Real Application Clusters. Executing a SQL statement in parallel will consume more of the machine resources—CPU, memory, and disk I/O—but complete the overall task faster.

Parallelism affects the amount of memory and CPU resources used to execute a given task in a fairly linear fashion—the more parallel processes used, the more resources consumed for the composite task. Each parallel execution process has a Program Global Area (PGA) that consumes memory and performs work. Each parallel execution process takes its own slice of CPU, but more parallel processes can reduce the total amount of time spent on disk I/O, which is the place in which bottlenecks can most frequently appear.

Two types of parallelism are possible within an Oracle Database:

Block-range parallelism
: Driven by ranges of database blocks

Partition-based parallelism
: Driven by the number of partitions or subpartitions involved in the operation

The following sections describe these types of parallelism.

Block-Range Parallelism

Oracle has featured the ability to dynamically parallelize table scans and a variety of scan-based functions since Oracle7. This parallelism is based on the notion of *block ranges*, in which the Oracle server understands that each table contains a set of data blocks that spans a defined range of data. Block-range parallelism is implemented by dynamically breaking a table into pieces, each of which is a range of blocks, and then using multiple processes to work on these pieces in parallel. Oracle's implementation of block-range parallelism is unique in that it doesn't require physically partitioned tables to achieve parallelism.

With block-range parallelism, the client session that issues the SQL statement transparently becomes the parallel execution coordinator, dynamically determining block ranges and assigning them to a set of parallel execution (PE) processes. Once a PE process has completed an assigned block range, it returns to the coordinator for more work. Not all I/O occurs at the same rate, so some PE processes may process more blocks than others. This notion of "stealing work" allows all processes to participate fully in the task, providing maximum leverage of the machine resources.

Block-range parallelism scales linearly based on the number of PE processes if there are adequate hardware resources. The key to achieving scalability with parallelism lies in hardware basics. Each PE process runs on a CPU and requests I/O to a device. If you have enough CPU processing power reading enough disks, parallelism will scale. If the system encounters a bottleneck on one of these resources, scalability will suffer. For example, four CPU cores reading two disks will not scale much beyond the two-way scalability of the disks and may even sink below this level if the additional CPUs cause contention for the disks. Similarly, two CPU cores reading 20 disks will not scale to a 20-fold performance improvement. The system hardware must be balanced for parallelism to scale.

Historically, most large systems had far more disks than CPU cores. In these systems, parallelism results in a randomization of I/O across the I/O subsystem. This is useful for concurrent access to data as PE processes for different users read from different disks at different times, resulting in I/O that is distributed across the available disks.

A useful analogy for dynamic parallelism is eating a pie. The pie is the set of blocks to be read for the operation, and the goal is to eat the pie as quickly as possible using a certain number of people. Oracle serves the pie in helpings, and when a person finishes his first helping, they can come back for more. Not everyone eats at the same rate, so some people will consume more pie than others. While this approach in the real world is somewhat unfair, it's a good model for parallelism because if everyone is eating all the time, the pie will be consumed more quickly. The alternative is to give each person an equal serving and wait for the slower eaters to finish.

Figure 7-3 illustrates the splitting of a set of blocks into ranges.

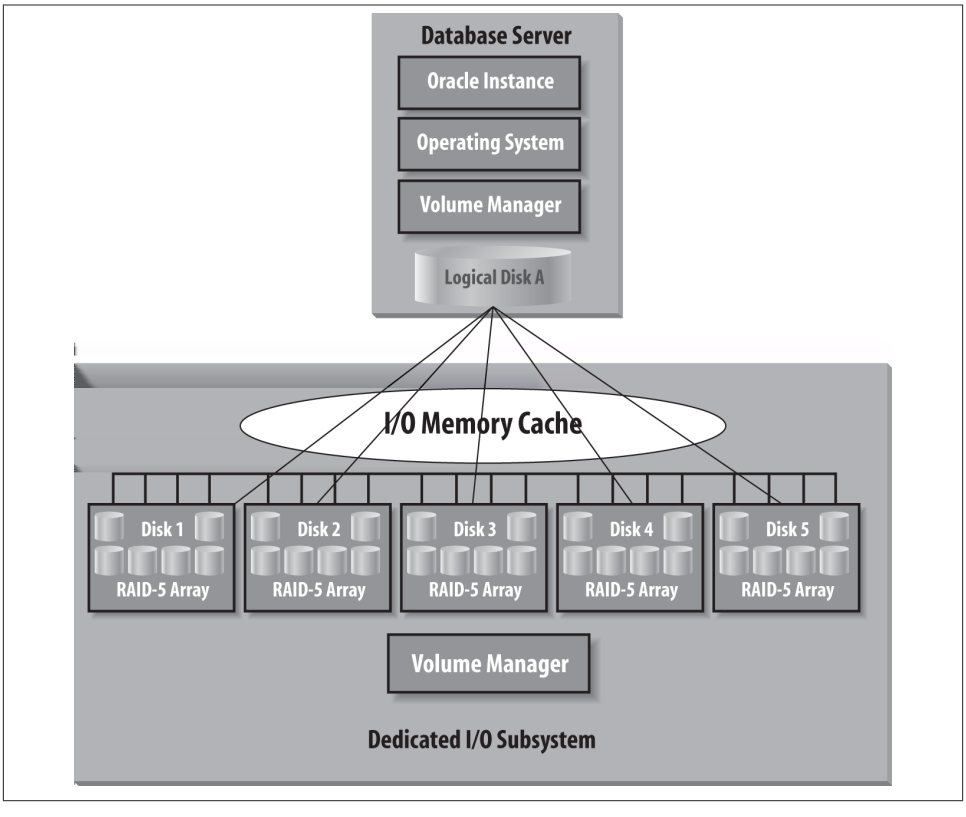


Figure 7-3. Dynamic block-range parallelism

It is worth noting here that in engineered systems such as the Oracle Exadata Database Machine, the number of CPU cores in the Database Server nodes is almost equal the number of disk drives in the Exadata Storage Server cells. Furthermore, there are additional CPU cores present in the Storage Server cells that help process the workload.

Parallelism for Tables and Partitions of Tables

Since *partitioned tables* were introduced in Oracle8, an operation may involve one, some, or all of the partitions of a partitioned table. There is essentially no difference in how block-range parallelism dynamically splits the set of blocks to be read for a regular table as opposed to a partitioned table. Once the optimizer has determined which partitions should be accessed for the operation, all the blocks of all partitions involved are treated as a pool to be broken into ranges.

This assumption by the optimizer leads to a key consideration for using parallelism and partitioned tables. The degree of parallelism (i.e., the number of parallel execution processes used for the table as a whole) is applied to the set of partitions that will be used for an operation. The optimizer will eliminate the use of partitions that do not contain data an operation will use. For instance, if one of the partitions for a table contains ID numbers below 1,000, and if a query requests ID numbers between 1,100 and 5,000, the optimizer understands that this query will not access this partition.

The Oracle Database provides a variety of partitioning techniques, including various combinations called composite partitioning, in the Partitioning Option. We described partitioning techniques in Chapter 4.

As you might expect, when your queries will use partition elimination or pruning and parallelism, the partitions should be striped over a sufficient number of drives to scale effectively. This will ensure scalability regardless of the number of partitions accessed. As noted previously, such striping is provided by ASM today.

What Can Be Parallelized?

Oracle can parallelize far more than simple queries. Some of the operations that can be parallelized using block-range parallelism include the following:

- Tablespace creation
- Index creation and rebuilds
- Online index reorganizations and rebuilds
- Index-organized table reorganizations and movements
- Table creation, such as summary creation using `CREATE TABLE ... AS SELECT`
- Partition-maintenance operations, such as moving and splitting partitions
- Data loading
- Integrity constraints imposing
- Statistics gathering (automatically gathered since Oracle Database 10*g*)
- Backups and restores (including very large files since Oracle Database 11*g*)
- DML operations (`INSERT`, `UPDATE`, `DELETE`)
- Query processing operations
- OLAP aggregate (since Oracle Database 10*g*)

Oracle can also provide the benefits of parallelism to individual processing steps for queries. Some of the specific features of query processing that may be parallelized include:

- Table scans
- Nested loops
- Sort merge joins
- Hash joins
- Bitmap star joins
- Index scans
- Partition-wise joins
- Anti-joins (NOT IN)
- SELECT DISTINCT
- UNION and UNION ALL
- ORDER BY
- GROUP BY
- Aggregations
- Import
- User-defined functions

Degree of parallelism

An Oracle instance has a pool of parallel execution (PE) processes that are available to the database applications. Controlling the number of active PE processes was an important task in older Oracle Database releases; too many PE processes would overload the machine, leading to resource bottlenecks and performance degradation. A high degree of parallelism would also force full table scans, and this may or may not be appropriate. Figure 7-4 illustrates transparent parallelism within and between sets of PE processes.

Determining the optimal degree of parallelism in the presence of multiple users and varying workloads can be challenging. For example, a degree of 8 for a query would provide excellent performance for 1 or 2 users, but what if 20 users queried the same table? This scenario called for 160 PE processes (8 PEs for each of the 20 users), which could overload the machine.

Setting the degree to a lowest common denominator value (for example, 2) provided effective parallelism for higher user counts, but did not leverage resources fully when fewer users were active.

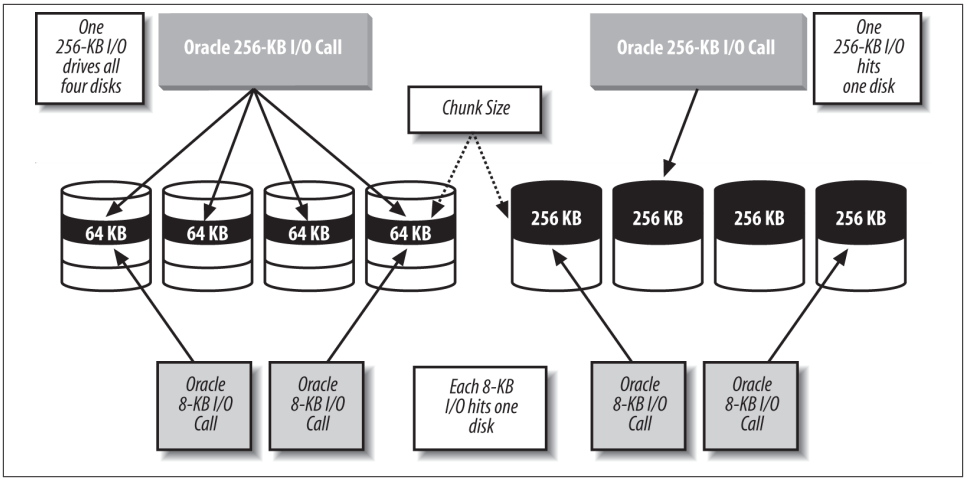


Figure 7-4. Intra-operation and inter-operation parallelism

Self-tuning adaptive parallelism

Oracle8*i* introduced the notion of *self-tuning adaptive parallelism*. This feature is turned on when the initialization parameter PARALLEL_DEGREE_POLICY is set to AUTO. Oracle automatically scales down parallelism as the system load increases and scales it back up as the load decreases. When an operation requests a degree of parallelism, Oracle will check the system load and lower the actual degree the operation uses to avoid overloading the system. As more users request parallel operations, the degree they receive will become lower and lower until operations are executing serially. If activity decreases, subsequent operations will be granted increasing degrees of parallelism. This adaptability frees the DBA from the difficult task of trying to determine the optimal degree of parallelism in the face of constant changes in workload.

Adaptive parallelism takes two factors into account in determining the degree of parallelism granted to an operation:

- System load.
- Parallelism resource limitations of the user's consumer group if the Database Resource Manager is active. (The Database Resource Manager is described later in this chapter.) This is important, because it means that adaptive parallelism respects resource plans if they're in place.

When automatic degree of parallelism is set as noted above, the Oracle Database will also queue SQL statements that require parallel execution if there are not enough PE server processes available. It will then execute those statements as needed resources do become available.

The database will also automatically decide whether an object being accessed should be held in the buffer cache (SGA) to speed parallel execution. Oracle refers to this as *in-memory parallel execution*. We describe the SGA and memory resources later in this chapter.

Partition-Based Parallelism

A small subset of Oracle's parallel functionality is based on the number of partitions or subpartitions accessed by the statement to be parallelized. For block-range parallelism, the piece of data each PE process works on is a range of blocks. For partition-based parallelism, the pieces of data that drive parallelism are partitions or subpartitions of a table. The operations in which parallelism is based on the number of partitions or subpartitions include the following:

- Updates and deletes
- Index scans
- Index creation and rebuilds on partitioned tables

Parallelism for partitions and subpartitions of a table

Oracle supports parallel Data Manipulation Language (DML), or the ability to execute INSERT, UPDATE, and DELETE statements in parallel. This type of parallelism improves the performance of large bulk operations (for example, an update to all the rows of a very large table).

Since Oracle8*i*, the degree of parallelism for updates and deletes is tied to the number of partitions or subpartitions involved. A table with 12 partitions (for example, one partition for each month of the year) can have a maximum number of 12 PEs for an update or delete. An update to only one month of data would have no parallelism because it involves only one partition. If the table were created using Oracle's composite partitioning (for example, with 4 hash subpartitions by PRODUCT_ID within each month), the maximum degree of parallelism for the entire table would be 48, or 12 partitions with 4 subpartitions each. An update to one month of data could have a degree of 4 because each month contains 4 hash subpartitions. If the table is not partitioned, Oracle cannot perform updates or deletes in parallel.

Oracle can also execute index creation, index rebuilds, and index scans for partitioned indexes in parallel using the same semantics as parallel DML: one PE process per partition or subpartition of the index.

Fast full-index scans for nonpartitioned tables

People often assume that the Oracle Database can parallelize index scans only if the target index is partitioned. However, Oracle has the ability to perform parallel index scans on nonpartitioned indexes for certain cases. If the index scan operation were

"unbounded," meaning that the entire index was going to be accessed to satisfy the query, then Oracle would use block-range parallelism to access the entire index in parallel. While Oracle can perform index scans for nonpartitioned indexes, this feature applies to a narrow set of queries. Partition-based index scans apply to a much broader range of queries.

Parallel insert for nonpartitioned and partitioned tables

Oracle can execute an INSERT statement of the form INSERT INTO *tableX* SELECT . . . FROM *tableY* in parallel for nonpartitioned and partitioned tables. Oracle uses a set of PE processes executing block-range parallelism for the SELECT portion of the INSERT statement. These PE processes pass the rows to a second set of PE processes, which insert the rows into the target table. The target table can be a nonpartitioned or partitioned table. Parallelism for an insert is not exactly block-range or partition-based.

Oracle and Memory Resources

Accessing data in memory is much faster than accessing data on disk. An Oracle instance uses the database server's memory resources to cache the information accessed to improve performance. Oracle utilizes an area of shared memory called the System Global Area (SGA) and a private memory area for each server process called the Program Global Area (PGA).

Prior to Oracle9*i*, you could only specify the size for the SGA or any of its components —database buffer cache, shared pool, or large pool—in the initialization file, and the size of these memory allocations could not be changed without shutting down and restarting the instance. Oracle9*i* enabled dynamic resizing of these pools based on a minimum memory allocation called a *granule*. Oracle Database 10*g* and later releases can automatically manage shared memory. Oracle Database 11*g* added automatic memory management of the SGA and PGA.

Exhausting a database server's supply of memory will cause poor performance. If you are running an older release of Oracle, you should gauge the size of the various memory areas Oracle uses or add more memory to the machine to prevent a memory deficit from occurring. What constitutes the right size for the various areas is a function of your application behavior, the data it uses, and your performance requirements.

How Oracle Uses the System Global Area

Oracle uses the SGA for the following operations:

- Caching of database blocks containing table and index data in the database buffer cache
- Caching of parsed and optimized SQL statements, stored procedures, and data dictionary information in the shared pool

- Buffering of redo log entries in the redo log buffer before they're written to disk

In versions prior to Oracle 9*i*, the amount of memory allocated to each of these areas within the SGA was determined at instance startup using initialization parameters and could not be altered without restarting the instance. The majority of tuning efforts focused on the database buffer cache and the shared pool.

Automatic sizing for the SGA

Oracle Database 10*g* eliminated manual tuning of SGA pools with automatic sizing for the SGA. Using automatic shared memory management, the database automatically allocates memory for the following SGA pools: database buffer cache, shared pool, large pool, Java pool, and Streams pool. You have to specify only the total amount of memory required by setting the SGA_TARGET initialization parameter.

Since Oracle Database 10*g*, the database proactively monitors the memory requirements for each pool and dynamically reallocates memory when appropriate. You can also specify the minimum amount of memory for any of the SGA pools while using automatic SGA sizing using the following initialization parameters: DB_CACHE_SIZE, SHARED_POOL_SIZE, LARGE_POOL_SIZE, JAVA_POOL_SIZE, and STREAMS_POOL_SIZE. A few of the SGA pools are still manually managed by specifying parameters such as LOG_BUFFER, DB_KEEP_CACHE_SIZE, and DB_RECYCLE_CACHE_SIZE.

The database buffer cache

If you decide to disable SGA_TARGET by setting it to 0, you will need to manually set initialization parameters for the memory pools. For the database buffer cache, you would assess the percentage of the database blocks requested by users read from the cache versus from the disk. This percentage is termed the *hit ratio*. If query response times are too high and the hit ratio is lower than 90 percent (as a rule of thumb), increasing the value of the initialization parameter DB_CACHE_SIZE can improve performance.

You can use Oracle Enterprise Manager to get information about the cache hit ratio.

It is tempting to assume that continually increasing the size of the database buffer cache will translate into better performance. However, this is true only if the database blocks in the cache are actually being reused. Most OLTP systems have a relatively small set of core tables that are heavily used (for example, lookup tables for things such as valid codes). The rest of the I/O tends to be random, accessing a row or two in various database blocks in the course of the transaction. Because of this, having a larger buffer cache may not contribute to performance since there isn't much reuse of data blocks occurring.

In addition, not all operations read from the database buffer cache. For example, large full table scans are limited to a small number of buffers to avoid adversely impacting other users by dominating the cache. If your application performs a lot of table scans, increasing the buffer cache may not help performance because the cache will not contain the needed data blocks. Parallel table scans completely bypass the buffer cache and pass rows directly to the requesting user process, as do Smart Scan operations on Exadata. As with most performance issues, your understanding of how your application is actually using your data is the key that will help guide your database buffer cache tuning.

The shared pool

The shared pool is used at several points during the execution of every operation that occurs in the Oracle Database. For example, the shared pool is accessed to cache the SQL sent to the database and for the data dictionary information required to execute the SQL. Because of its central role in database operations, a shared pool that is too small may have a greater impact on performance than a database buffer cache that is too small. If the requested database block isn't in the database buffer cache, Oracle will perform an I/O to retrieve it, resulting in a one-time performance hit.

A shared pool that is too small will cause poor performance for a variety of reasons, affecting all users. These reasons include the following:

- Not enough data dictionary information can be cached, resulting in frequent disk access to query and update the data dictionary.
- Not enough SQL can be cached, leading to memory churn or the flushing of useful statements to make room for incoming statements. A well-designed application issues the same statements repeatedly. If there isn't enough room to cache all the SQL the application uses, the same statements get parsed, cached, and flushed over and over, wasting valuable CPU resources and adding overhead to every transaction.
- Not enough stored procedures can be cached, leading to similar memory churn and performance issues for the program logic stored and executed in the database.

If you are manually managing the shared pool and you've diagnosed which of these problems is occurring, the solution is fairly simple: increase the size of the shared pool using the SHARED_POOL_SIZE initialization parameter. For more information about examining shared pool activity to identify problems, see the appropriate *Oracle Performance Tuning Guide*, as well as the other books on this topic.

The redo log buffer

While the redo log buffer consumes a very small amount of memory in the SGA relative to the Oracle Database buffer cache and the shared pool, it is critical for performance. Transactions performing changes to the data in the database write their redo information to the redo log buffer in memory. The redo log buffer is flushed to the redo logs on

disk when a transaction is committed (normally) or when the redo log buffer is one-third full. Oracle "fences" off the portion of the redo log buffer that's being flushed to disk to make sure that its contents aren't changed until the information is safely on disk. Transactions can continue to write redo information to the rest of the redo log buffer (the portion that isn't being written to disk and therefore isn't fenced off by Oracle). In a busy database, transactions may generate enough redo to fill the remaining unfenced portion of the redo log buffer before the I/O to the disks for the fenced area of the redo log buffer is complete. If this happens, the transactions will have to wait for the I/O to complete because there is no more space in the redo log buffer. This situation can impact performance. The statistic "redo buffer allocation retries" can be used to understand this situation.

You monitor these statistics over a period of time to gain insight into the trend. The values at one point in time reflect the cumulative totals since the instance was started and aren't necessarily meaningful as a single data point. Note that this is true for all statistics used for performance tuning. Ideally, the value of "redo buffer allocation retries" should be close to 0. If you observe the value rising during the monitoring period, you would increase the size of the redo log buffer by resetting the LOG_BUFFER initialization parameter.

Query results caching

Oracle caches database and index blocks, eliminating the need to perform resource-intensive disk reads. Oracle also caches SQL plans, eliminating the need to reparse and optimize queries. But prior to Oracle Database 11*g*, a cached SQL plan would still have to execute and assemble a result set for repeated queries.

Oracle Database 11*g* and newer database releases cache the completed result set in the shared pool. This functionality means that a repeated query requesting the same result set can simply take that result set completely from memory. Since the result sets have to be the same for this feature to work, the query results cache has the biggest impact on situations like web page serving, where the same page is being retrieved repeatedly. This feature also works on the results of PL/SQL functions.

Oracle Database 11*g* and newer database releases also have the ability to cache query result sets on the client, while automatically keeping the result set consistent with any changes that could affect it. This feature gives the performance benefits of query result set caching on the server while eliminating network roundtrips as an added benefit.

How Oracle Uses the Program Global Area

Each server has a Program Global Area (PGA), which is a private memory area that contains information about the work the server process is performing. There is one PGA for each server process. The total amount of memory used for all the PGAs is a function of the number of server processes active as part of the Oracle instance. The

larger the number of users, the higher the number of server processes and the larger the amount of memory used for their associated PGAs. Using shared servers reduces total memory consumption for PGAs because it reduces the number of server processes.

The PGA consists of a working memory area for things such as temporary variables used by the server process, memory for information about the SQL the server process is executing, and memory for sorting rows as part of SQL execution. The initial size of the PGA's working memory area for variables, known as *stack space*, cannot be directly controlled because it's predetermined based on the operating system you are using for your database server. The other areas within the PGA can be controlled as described in the following sections.

Memory for SQL statements

When a server process executes a SQL statement for a user, the server process tracks the session-specific details about the SQL statement and the progress by executing it in a piece of memory in the PGA called a *private SQL area*, also known as a *cursor*. This area should not be confused with the shared SQL area within the shared pool. The shared SQL area contains shareable details for the SQL statement, such as the optimization plan. Optimization plans are discussed in Chapter 4.

The private SQL area contains the session-specific information about the execution of the SQL statement within the session, such as the number of rows retrieved so far. Once a SQL statement has been processed, its private SQL area can be reused by another SQL statement. If the application reissues the SQL statement whose private SQL area was reused, the private SQL area will have to be reinitialized.

Each time a new SQL statement is received, its shared SQL area must be located (or, if not located, loaded) in the shared pool. Similarly, the SQL statement's private SQL area must be located in the PGA or, if it isn't located, reinitialized by the server process. The reinitialization process is relatively expensive in terms of CPU resources.

A server process with a PGA that can contain a higher number of distinct private SQL areas will spend less time reinitializing private SQL areas for incoming SQL statements. If the server process doesn't have to reuse an existing private SQL area to accommodate a new statement, the private SQL area for the original statement can be kept intact. Although similar to a larger shared pool, a larger PGA avoids memory churn within the private SQL areas. Reduced private SQL area reuse, in turn, reduces the associated CPU consumption, increasing performance. There is, of course, a trade-off between allocating memory in the PGA for SQL and overall performance.

OLTP systems typically have a "working set" of SQL statements that each user submits. For example, a user who enters car rental reservations uses the same forms in the application repeatedly. Performance will be improved if the user's server process has enough memory in the PGA to cache the SQL those forms issue. Application developers

should also take care to write their SQL statements so that they can be easily reused, by specifying bind variables instead of different hardcoded values in their SQL statements.

Memory for sorting within the PGA

Each server process uses memory in its PGA for sorting rows before returning them to the user. If the memory allocated for sorting is insufficient to hold all the rows that need to be sorted, the server process sorts the rows in multiple passes called *runs*. The intermediate runs are written to the temporary tablespace of the user, which reduces sort performance because it involves disk I/O.

Sizing the sort area of the PGA was a critical tuning point in Oracle Database releases prior to Oracle Database 10*g*. A sort area that was too small for the typical amount of data requiring sorting would result in temporary tablespace disk I/O and reduced performance. A sort area that was significantly larger than necessary would waste memory.

Since Oracle Database 10*g*, Oracle provides automatic sizing for the PGA. By default, this memory management is enabled, and sizing for PGA work areas is based on 20 percent of the SGA memory size. By using automatic sizing for the PGA, you eliminate the need to size individual portions of the PGA, such as SORT_AREA_SIZE.

Oracle Database 11*g* introduced automatic memory management that spans both the SGA and the PGA. By setting a single MEMORY_TARGET initialization parameter (given that the PGA size can be based on a percentage of the SGA memory size), the PGA and SGA will be automatically set to appropriate initial values. Oracle then tunes memory for optimal SGA and PGA performance on an ongoing basis.

Oracle and CPU Resources

The Oracle Database shares the CPU(s) with all other software running on the server. If there is a shortage of CPU power, reducing Oracle or non-Oracle CPU consumption will improve the performance of all processes running on the server.

If all the CPUs in a machine are busy, the processes line up and wait for a turn to use the CPU. This is called a *run queue* because processes are waiting to run on a CPU. The busier the CPUs get, the longer processes can spend in this queue. A process in the queue isn't doing any work, so as the run queue gets longer, response times degrade.

You can use the standard monitoring tools for your particular operating system to check the CPU utilization for that machine.

Tuning CPU usage is essentially an exercise in tuning individual tasks by reducing the number of commands required to accomplish the tasks and/or reducing the overall number of tasks to be performed. You can do this tuning through workload balancing,

SQL tuning, or improved application design. This type of tuning requires insight into what these tasks are and how they're being executed.

As mentioned earlier, an in-depth discussion of all the various tuning points for an Oracle Database is beyond the scope of this book. However, there is a set of common tasks that typically result in excess CPU consumption. Some of the usual suspects to examine if you encounter a CPU resource shortage on your database server include the following:

Bad SQL

Poorly written SQL is the number one cause of performance problems. The Oracle Database attempts to optimally execute the SQL it receives from clients. If the SQL contained in the client applications and sent to the database is written so that the best optimization plan Oracle can identify is still inefficient, Oracle will consume more resources than necessary to execute the SQL. Tuning SQL can be a complex and time-consuming process because it requires an in-depth understanding of how Oracle works and what the application is trying to do. Initial examinations can reveal flaws in the underlying database design, leading to changes in table structures, additional indexes, and so on. Changing the SQL required retesting and a subsequent redeployment of the application—until Oracle Database 10*g*.

Oracle Database 10*g* introduced the SQL Tuning Advisor, a tool that can not only recognize poorly written SQL, but also create an optimizer plan to circumvent the problem and replace the standard optimization plan with the improved plan. With this capability, you can improve the performance of poorly written SQL without changing any code in the application.

Since Oracle Database 11*g*, the Oracle Database can automatically spot the SQL queries with the largest loads and automatically create SQL profiles to improve their performance, if appropriate. This process can also result in advice on new indexes that could improve the performance of these statements.

These newer releases of the Oracle Database also track changes in execution plans for SQL statements. The optimizer can maintain the history of execution plans, and when a new plan is detected, the optimizer uses the old plan and evaluates the performance of the new plan. Once the optimizer verifies that the new plan can deliver the same performance, the old plan is replaced. This feature does not directly relate to bad SQL, but rather to the occasional effects of plan changes, which can result in unplanned performance degradation.

Excessive parsing

As we discussed in the section "Memory for SQL statements" on page 195, Oracle must parse every SQL statement before it's processed. Parsing is very CPU-intensive, involving a lot of data dictionary lookups to check that all the tables and columns referenced are valid. Complex algorithms and calculations estimate the costs of the various optimizer plans possible for the statement to select the optimal

plan. If your application isn't using bind variables (discussed in Chapter 9), the Oracle Database will have to parse every statement it receives. This excessive and unnecessary parsing is one of the leading causes of performance degradation. Another common cause is a shared pool that's too small, as discussed previously in the section "The shared pool." Keep in mind that you can avoid the creation of execution plans by using stored outlines, as described in Chapter 4. And, since Oracle9*i*, you also have the ability to edit the hints that make up a stored outline. As described earlier, Oracle Database 11*g* and newer Oracle Database releases include the ability to cache complete result sets, which can minimize the impact of repeated execution of identical queries.

Database workload
: If your application is well designed and your database is operating at optimal efficiency, you may experience a shortage of CPU resources for the simple reason that your server doesn't have enough CPU power to perform all the work it's being asked to do. This shortage may be due to the workload for one database (if the machine is a dedicated database server) or to the combined workload of multiple databases running on the server. Underestimating the amount of CPU resources required is a chronic problem in capacity planning. Unfortunately, accurate estimates of the CPU resources required for a certain level of activity demands detailed insight into the amount of CPU power each transaction will consume and how many transactions per minute or second the system will process, both at peak and average workloads. Most organizations don't have the time or resources for the system analysis and prototyping required to answer these questions. The common solution is to simply add more CPU resources to the machine until the problem goes away.

Nondatabase workload
: Not all organizations dedicate an entire machine to an Oracle Database to ensure that all CPU resources are available for that database. Use operating system utilities to identify the top CPU consumers on the machine. You may find that non-Oracle processes are consuming the bulk of the CPU resources and adversely impacting Oracle Database performance.

Performance Tuning Basics

There are three basic steps to understanding how to address performance issues with your Oracle Database:

1. Define performance and performance problems.
2. Check the performance of the Oracle Database software.
3. Check the overall performance of the server and storage.

Database Administrators generally tune the Oracle Database proactively using Oracle's real-time diagnostics capabilities, reactively (especially where transient performance problems are occurring), or most commonly using a combination of both techniques.

Defining Performance and Performance Problems

The first step in performance tuning is to determine if there actually *is* a performance problem. In the previous section, we mentioned the concept of poor performance and how users often are the first to recognize it. But what exactly is poor performance?

Poor performance is inevitably the result of disappointment—a user feels that the system is not performing as expected. Consequently, you must first evaluate how real these expectations are in the first place.

Disappointment, in part, stems from expectations. Because of this, inconsistent performance is often identified as poor performance. If an operation runs fast sometimes and slower at other times, the slower executions stand out. This simple truism is why Oracle offers options such as fixed execution plans with stored outlines.

If expectations are realistic—for example, a scenario where performance has degraded from a previous level and the business is affected—you then need to identify which of the system's components are causing the problems. You must refine a general statement like "the system is too slow" to identify which types of operations are too slow, what constitutes "too slow," and when these operations are slowing down. For example, the problem may occur only on specific transactions and at specific times of day, or all transactions and reports may be performing below the users' expectations.

Once you've defined the performance expected from your system, you can begin to try to determine where the performance problem lies. Performance problems occur when there is a greater demand for a particular resource than the resources available to service that demand, and the system slows down while applications wait to share the resource.

Monitoring and Tuning the Oracle Database for Performance

The first place you'll likely begin looking for resource bottlenecks is in the Oracle Database software using Oracle Enterprise Manager (introduced in Chapter 5) to identify less than optimal use of Oracle's internal resources. Bottlenecks within your Oracle Database result in sessions waiting for resources, and performance tuning is aimed at removing these bottlenecks. The Enterprise Manager Diagnostics Pack includes three important components used in performance tuning: the Automatic Workload Repository (AWR), the Active Session History (ASH), and the Automatic Database Diagnostic Monitor (ADDM).

The AWR captures and stores information about resource utilization by Oracle workloads. By default, statistics are captured every 60 minutes and are stored for 8 days. The AWR helps the Oracle Database identify potential performance issues by comparing workloads over time. It also acts as the foundation for many of the manageability features introduced since Oracle Database 10*g*, such as the ADDM.

AWR gathers statistics from dynamic performance views. The views were sometimes directly accessed by DBAs in the past, but most DBAs simply use Enterprise Manager to manage the database today. The performance views have names that begin with *V$* or, if global views (for all nodes in a Real Application Clusters, or RAC, database), begin with *GV$*. For example, views that provide statistics useful in uncovering the sources of waits include:

V$SYSTEM_EVENT
: Provides aggregated, system-wide information about the resources for which the whole instance is waiting

V$SESSION_EVENT
: Provides cumulative list of events waited for in each session

V$SESSION
: Provides session information for each current session including the event currently or last waited for

ASH captures statistics for active sessions in memory that are then written into storage using AWR snapshot processing. Data collected includes the SQL identifier of a SQL statement, the object, file, and block numbers, wait event identifier and event parameters, session identifier and serial number, module and action name, session client identifier, and service hash identifier. You can run ASH reports from the performance page in Oracle Enterprise Manager. The reports can help you determine what is causing spikes in Oracle Database activity and can indicate the top events (consuming CPU and wait for CPU), load profile, top SQL, top sessions (waiting for wait events), top Database objects, files, and latches, and activity over time.

ASH reports, in Enterprise Manager 12*c*, added three new reports. ASH analytics enables exploration of different session performance metrics at different points in time through the creation of filters. Support for Compare Period ADDM and Real-Time ADDM was also added.

Oracle's ADDM automatically identifies and reports on resource bottlenecks, such as CPU contention, locking issues, or poor performance from specific SQL statements. Since Oracle Database 11*g*, ADDM can also perform analysis on clusters. Real-time monitoring of user activity, instance activity, and host activity is supported.

Enterprise Manager Cloud Control displays the results of ADDM monitoring and provides both high-level and detailed views of resource utilization for Oracle Databases. The dashboards can help give a quick indication of the cause of performance problems.

For example, the Top SQL section of the Top Activity Oracle performance page shows high load SQL statements by wait class and provides a SQL ID link to enable detailed viewing of a SQL statement in question. Thresholds on various metrics being gathered can be set such that the dashboard will clearly indicate when a particular resource is nearing a critical usage level and alerts are then sent as appropriate.

ADDM also sends Oracle Database change and management recommendations to Enterprise Manager that DBAs can take action upon. These include recommending Oracle Database initialization parameter changes, using hash partitioning on a table or index, using automatic segment space management (ASSM), using the cache option or bind variables in applications, making hardware changes (such as CPUs and/or I/O configuration), and running additional advisors including the SQL Tuning Advisor and Segment Advisor.

Enterprise Manager offers a number of optional Packs that include these and other advisors that we introduced in Chapter 5. The performance-related advisors can give you further suggestions on how to tune your applications or optimize performance in the Oracle Database.

Two of the key advisors for performance tuning, the SQL Tuning and SQL Access Advisor, are included in the Oracle Database Tuning Pack. These leverage information on CPU and I/O consumption captured in the AWR and are used to make tuning recommendations where high-load SQL statements are indicated by the ADDM. The SQL Tuning Advisor makes recommendations on how to better restructure SQL statements and the creation of SQL profiles. The SQL Access Advisor identifies optimal access paths through SQL profiling, and determines if the addition of indexes, materialized views, or other database structures would be beneficial. Using these two advisors together, the DBA can determine whether these changes to the high-load SQL statements would improve efficiency.

Proactive Oracle Database performance tuning relies on DBAs reviewing ADDM findings and responding to ADDM recommendations. Techniques commonly used include real-time monitoring of the Oracle Database and response to alerts generated.

Reactive tuning also leverages ADDM, but often includes investigating ASH reports and setting up and preserving database baselines for generating AWR Compare reports to better understand database performance changes over time when transient problems occur. The baselines can be established for fixed times, moving windows, and can serve as templates.

Prior to making changes to your production Oracle Database, you might want to first test the changes on a test version of the Oracle Database to understand potential per-

formance implications. DBAs often use Real Application Testing (RAT) and the Database Replay component to capture the workload on the production Oracle Database system, and then replay the workload on the test system with the same timing and concurrency characteristics. The SQL Performance Analyzer (SPA) in this Pack identifies the SQL statements that have slowed, stayed the same, or improved on the test system.

Using the Oracle Database Resource Manager

Oracle can help you manage resources and how they are allocated before performance becomes an issue. For example, the Oracle *Database Resource Manager (DRM)* works by leveraging consumer groups you've identified and enables you to place limits on the amount of computer resources that can be used by that group.

Normally, the limits imposed by Database Resource Manager consumer groups are only imposed when there is too much demand for a resource, such as more demand for CPU than is available. This approach makes sense, since there is no reason to leave CPU cycles unused. Since Oracle Database 11*g*, you can specify a maximum amount of CPU that is used in all cases, whether the CPU is oversubscribed or not.

Implementing the DRM ensures that one group or member of a group does not end up using an excessive amount of any one resource, as well as acting to deliver guaranteed service levels for different sets of users. You can create DRM hierarchies in which you specify the amount of resources for groups within groups.

The following DRM features can be combined to protect against poor performance:

Predicting resource utilization
: The DRM uses the query optimizer cost computations to predict the amount of resources that a given query will take and the query execution time. Note that, by default, the query optimizer uses a CPU + I/O cost model since Oracle Database 10*g*.

Switching consumer groups
: The DRM can switch consumer groups dynamically. You might want to give a particular consumer group a high allocation of CPU resources. But if a single query from that group looks as if it will take up too many CPU resources and impact the overall performance of the machine, the consumer group can be switched to another group that has a smaller CPU allocation—for example, a consumer group designed for batch operations.

Limiting number of connections
: The DRM can limit the number of connections for any particular consumer group. If the limit on connections for a group has been reached and another connection request comes in, the connection request is queued until an existing connection is freed. By limiting the overall number of connections for a consumer group, you can place some rough limits on the overall resources that particular group might require.

Since Oracle Database 11*g*, the Oracle Database is installed with a default DRM plan. The default plan is designed to limit the amount of resources used by automated maintenance tasks such as optimizer statistics gathering, the Segment Advisor, and the SQL Tuning Advisor.

The DRM is monitored through the Enterprise Manager Performance page. Charts illustrate resources used by each consumer group.

Additional Monitoring and Tuning Available for Oracle Exadata

Tuning of Oracle Databases on Exadata has many similarities to tuning Oracle on any platform. The AWR repository, ADDM, and SQL Tuning and SQL Access advisors all provide important information. However, when using Enterprise Manager with Exadata, there are additional capabilities provided since the Oracle Database Performance page in Enterprise Manager is Exadata-aware. It is possible to examine Exadata Storage Server cell offloads, smart scan activity, and I/O activity of all databases sharing the same storage. Explain plans show how Exadata Storage is helping to resolve queries. The Exadata System Health page presents an overview of system component availability.

Exadata Storage Server cells in the Oracle Exadata Database Machine are monitored as if a single target when using Enterprise Manager 12*c*. On the Exadata Storage Cell Grid page, you can view I/O load, CPU utilization, InfiniBand network utilization, and average response time across the cells. You can also view the status of the Storage Server software and the *I/O Resource Manager (IORM)*.

IORM is an important extension to the Database Resource Manager for enabling Exadata to meet service level agreements (SLAs) by managing I/O for different classes of workloads (e.g., interactive and reporting) and when multiple Oracle Databases are deployed on the same Exadata system. DBAs can define inter- and intra-database I/O bandwidth in addition to ensure that no database is starved for I/O. Database consumer groups can be allocated by percentage of available I/O bandwidth that is available. Inter-database I/O is managed within the cells by IORM. Intra-database consumer group I/O allocations are managed at the database server.

Quality of Service (QoS) Management and QoS Management Memory Guard are also supported on Exadata. QoS Management enables DBAs to set policies where critical business objectives can be assured of being met (at the expense of less critical objectives)

and monitors response times of applications and relieving bottlenecks. QoS Management Memory Guard determines if nodes are about to fail due to memory overcommitment and prevents additional new connections from taking place.

A Final Note on Performance Tuning

Performance has real-world business implications. Whenever you attempt to address performance problems, you must make sure to carefully monitor the areas that you are attempting to improve, both before and after your changes. Important baseline data gathered by the AWR includes application, database, operating system, disk I/O, and network statistics.

You should use a systematic approach to both discovering the source of a performance problem and implementing the appropriate solution. This approach calls for establishing baselines for resource usage and response time before making any changes, and only making a small group of changes before reexamining the performance in the changed environment. It might be tempting to simply try to fix a problem without taking a measured approach, but this tactic will usually lead to additional problems down the road.

CHAPTER 8
Oracle Multiuser Concurrency

Sharing data is at the center of all information systems. As systems provide higher and higher levels of functionality, we can sometimes forget that the ability to efficiently share data is the underlying governor of overall system performance. At the same time, database systems must protect the integrity of the data, as the value of that data is directly proportional to the correctness of the data. Database systems must protect data integrity, while still providing high levels of performance for multiuser access. These two forces sometimes conflict and shape some of the core technology in any database system.

Data integrity must always come first. As Ken Jacobs, fondly known as Dr. DBA in the Oracle community, put it in his classic paper entitled "Transaction Control and Oracle7," a multiuser database must be able to handle concurrently executing transactions in a way that "ensure(s) predictable and reproducible results." This goal is the core issue of data integrity, which, in turn, is the foundation of any database system.

When multiple users access the same data, there is always the possibility that one user's changes to a specific piece of data will be unwittingly overwritten by another user's changes. If this situation occurs, the accuracy of the information in the database is compromised, and that can render the data useless or, even worse, misleading. At the same time, the techniques used to prevent this type of loss can dramatically reduce the performance of an application system, as users wait for other users to complete their work before continuing. These techniques act like a traffic signal, so you can't solve this type of performance problem by increasing the resources available to the database. The problem isn't due to a lack of horsepower—it's caused by a red light.

Although concurrency issues are central to the success of applications, they are some of the most difficult problems to predict because they can stem from such complex interactive situations. The difficulties posed by concurrent access increase as the number of concurrent users increases. Even a robust debugging and testing environment may fail to detect problems created by concurrent access, since these problems are created by large numbers of users who may not be available in a test environment. Concurrency

problems can also pop up as user access patterns change throughout the life of an application.

If problems raised by concurrent access aren't properly handled by a database, developers may find themselves suffering in a number of ways. They will have to create their own customized solutions to these problems in their software, which will consume valuable development time. They will frequently find themselves adding code during the late stages of development and testing to work around the underlying deficiencies in their database systems, which can undercut the design and performance of the application. Worst of all, they may find themselves changing the optimal design of their data structures to compensate for weaknesses in the capabilities of the underlying database.

There is only one way to deal successfully with the issues raised by concurrent data access. The database that provides the access must implement strategies to transparently overcome the potential problems posed by concurrent access. Fortunately, Oracle has excellent methods for handling concurrent access. In fact, these methods have been one of the major differentiators of the Oracle Database for more than two decades.

This chapter describes the basics of concurrent data access and gives you an overview of the way that Oracle handles the issues raised by concurrent access. If you've worked with large database systems in the past and are familiar with concurrent user access, you might want to skip the first section of this chapter.

Basics of Concurrent Access

Before you can understand the problems posed by multiuser concurrent access to data, you need to understand the basic concepts that are used to identify and describe those potential concurrency issues.

Transactions

The *transaction* is the bedrock of data integrity in multiuser databases and the foundation of all concurrency schemes. A transaction is defined as a single indivisible piece of work that affects some data. All of the modifications made to data within a transaction are uniformly applied to a database with a COMMIT statement, or the data affected by the changes is uniformly returned to its initial state with a ROLLBACK statement. Once a transaction is committed, the changes made by that transaction become permanent and are made visible to other transactions and other users.

Transactions always occur over time, although most transactions occur over a very short period of time. Since the changes made by a transaction aren't official until the transaction is committed, each individual transaction must be isolated from the effects of other transactions. The mechanism used to enforce transaction isolation is the lock.

Locks

A database uses a system of *locks* to prevent transactions from interfering with each other. A lock prevents users from modifying data. Database systems use locks to keep one transaction from overwriting changes added by another transaction.

Figure 8-1 illustrates the potential problems that could occur if a system did not use locks. Transaction A reads a piece of data; Transaction B reads the same piece of data and commits a change to the data. When Transaction A commits the data, its change unwittingly overwrites the changes made by Transaction B, resulting in a loss of data integrity.

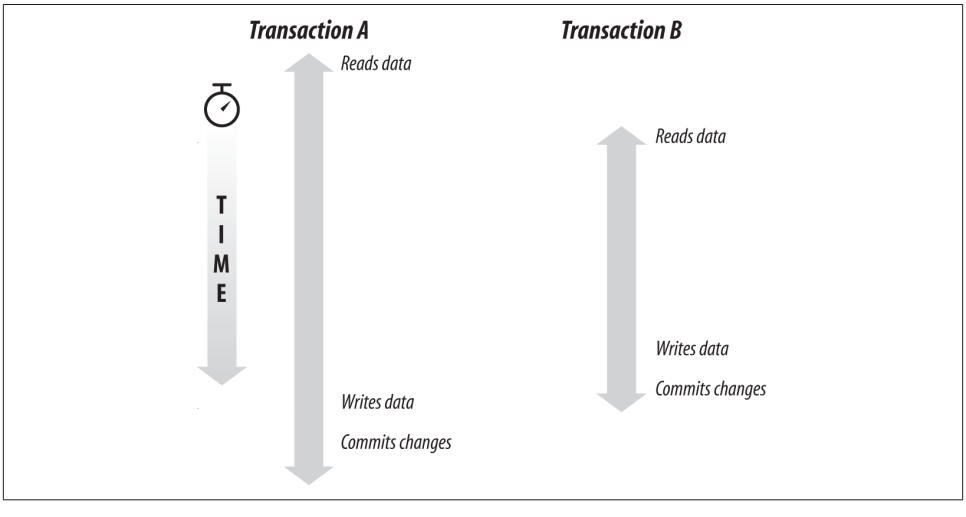


Figure 8-1. Transactions over time

Two types of locks are used to avoid this type of problem. The first is called a *write lock*, or an *exclusive lock*. An exclusive lock is applied and held while changes are made to data in the course of a transaction and released when the transaction is ended by either a COMMIT or a ROLLBACK statement. A write lock can be held by only one user at a time, so only one user at a time can change that data.

Some databases also use *read locks*, or *shared locks*. A read lock can be held by any number of users who are merely reading the data, since the same piece of data can be shared among many readers. However, a read lock prevents a write lock from being placed on the data, as the write lock is an exclusive lock. In Figure 8-1, if a read lock were placed on the data when Transaction A began, Transaction B would be able to read the same data but would be prevented from acquiring a write lock on the data until Transaction A ended.

Oracle uses locks when reading data only when a SQL operation specifically requests them with the FOR UPDATE clause in a SELECT statement, which causes write locks to be applied. You shouldn't use the FOR UPDATE clause routinely because it unduly increases the probability that readers will interfere with writers—a situation that normally never occurs with Oracle, as you will see shortly.

Concurrency and Contention

A system of locks enforcing isolation between concurrent users of data can lead to its own problems. As described in the example above, a single transaction can cause significant performance problems as the locks it places on the database prevent other transactions from completing. The interference caused by conflicting locks is called *contention*. More contention in a database slows response times and lowers the overall throughput.

In most other databases, increased concurrent access to data results in increased contention and decreased performance. Oracle's multiversion read concurrency scheme can greatly reduce contention, as later sections in this chapter will explain.

Integrity Problems

Some basic integrity problems can result if transaction isolation isn't properly enforced. Four of these problems are common to many databases:

Lost updates
: The most common type of integrity problem occurs when two writers are both changing the same piece of data, and one writer's changes overwrite the other writer's changes. This is the problem that exclusive locks are designed to prevent.

Dirty reads
: Occur when a database allows a transaction to read data that has been changed by another transaction but hasn't been committed yet. The changes made by the transaction may be rolled back, so the data read may turn out to be incorrect. Many databases allow dirty reads to avoid the contention caused by read locks.

Nonrepeatable reads
: Occur as a result of changes made by another transaction. One transaction makes a query based on a particular condition. After the data has been returned to the first transaction, but before the first transaction is complete, another transaction *changes* the data so that some of the previously retrieved data no longer satisfies the selection condition. If the query were repeated in the same transaction, it would return a different set of results, so any changes made on the basis of the original results may no longer be valid. Data that was read once can return different results if the data is read again later in the same transaction.

Phantom reads
: Also occur as a result of changes made by another transaction. One transaction makes a query based on a particular condition. After the data has been returned to the first transaction, but before the first transaction is complete, another transaction inserts into the database new rows that meet the selection criteria for the first transaction. If the first SQL statement in a transaction returned the number of rows that initially satisfied the selection criteria, and then performed an action on the rows that satisfied the selection criteria later in the transaction, the number of rows affected would be different from the initial number of rows indicated, based on the inclusion of new phantom rows.

Serialization

The goal of a complete concurrency solution is to provide the highest level of isolation between the actions of different users accessing the same data. As defined by the SQL92 standard, this highest level is called *serializable*. As the name implies, serializable transactions appear as though they have been executed in a series of distinct, ordered transactions. When one transaction begins, it is isolated from any changes that occur to its data from subsequent transactions.

To the user, a serializable transaction looks as though it has the exclusive use of the database for the duration of the transaction. Serializable transactions are predictable and reproducible, the two cardinal virtues of data integrity.

Of course, it's not trivial to have a database server support thousands of users while each one thinks he is the only one. But Oracle manages to pull off this quietly dramatic feat.

Oracle and Concurrent User Access

Oracle solves the problems created by concurrent access through a technology called *multiversion read consistency*, sometimes referred to as MVRC. Multiversion read consistency guarantees that a user sees a consistent view of the data she requests. If another user changes the underlying data during the query execution, Oracle maintains a version of the data as it existed at the time the query began. If there were transactions underway but uncommitted at the time the query began, Oracle will ensure that the query ignores the changes made by those transactions. The data returned to the query will reflect all committed transactions at the time the query started.

This feature has two dramatic effects on the way queries impact the database. First, Oracle doesn't place any locks on data for read operations. This means that a read operation will never block a write operation. Even where the database places a single lock on a single row as part of a read operation, that single lock can still cause contention in the database, especially since most database tables tend to concentrate update operations around a few "hot spots" of active data.

Second, a user gets a complete "snapshot" view of the data, accurate at the point in time that the query began. Other databases may reduce the amount of contention in the database by locking an individual row only while it's being read, rather than over the complete duration of the row's transaction. A row that's retrieved at the end of a result set may have been changed since the time the result set retrieval began. Because rows that will be read later in the execution of the query weren't locked, they could be changed by other users, which would result in an inconsistent view of the data.

Oracle's Isolation Levels

Oracle, like many other databases, uses the concept of *isolation levels* to describe how a transaction will interact with other transactions and how it will be isolated from other transactions. An isolation level is essentially a locking scheme implemented by the database that guarantees a certain type of transaction isolation.

An application programmer can set an isolation level at the session level (ALTER SESSION) or transaction level (SET TRANSACTION). More restrictive isolation levels will cause more potential contention, as well as delivering increased protection against data integrity problems.

Two basic isolation levels are used frequently within Oracle: READ COMMITTED and SERIALIZABLE. (A third level, READ ONLY, is described later in this section.) Both of these isolation levels create serializable database operations. The difference between the two levels is in the duration for which they enforce serializable operations:

READ COMMITTED
: Enforces serialization at the statement level. This means that every statement will get a consistent view of the data as it existed at the start of that statement. However, since a transaction can contain more than one statement, it's possible that nonrepeatable reads and phantom reads can occur within the context of the complete transaction. The READ COMMITTED isolation level is the default isolation level for Oracle.

SERIALIZABLE
: Enforces serialization at the transaction level. This means that every statement within a transaction will get the same consistent view of the data as it existed at the start of the transaction.

Because of their differing spans of control, these two isolation levels also react differently when they encounter a transaction that blocks their operation with an exclusive lock on a requested row. Once the lock has been released by the blocking transaction, an operation executing with the READ COMMITTED isolation level will simply retry the operation. Since this operation is concerned only with the state of data when the statement begins, this is a perfectly logical approach.

On the other hand, if the blocking transaction commits changes to the data, an operation executing with a SERIALIZABLE isolation level will return an error indicating that it cannot serialize operations. This error makes sense, because the blocking transaction will have changed the state of the data from the beginning of the SERIALIZABLE transaction, making it impossible to perform any more write operations on the changed rows. In this situation, an application programmer will have to add logic to his program to return to the start of the SERIALIZABLE transaction and begin it again.

There are step-by-step examples of concurrent access later in this chapter (in the section "Concurrent Access and Performance" on page 217) that illustrate the different ways in which Oracle responds to this type of problem.

One other isolation level is supported by Oracle: you can declare that a session or transaction has an isolation level of READ ONLY. As the name implies, this level explicitly prohibits any write operations and provides an accurate view of all the data at the time the transaction began.

Oracle Concurrency Features

Three features are used by Oracle to implement multiversion read consistency:

UNDO segments

UNDO segments are structures in the Oracle Database that store "undo" information for transactions in case of rollback. This information restores database rows to the state they were in before the transaction in question started. When a transaction starts changing some data in a block, it first writes the old image of the data to an UNDO segment. The information stored in an UNDO segment provides the information necessary to roll back a transaction and supports multiversion read consistency.

An UNDO segment is different from a redo log. The redo log is used to log all transactions to the database and recover the database in the event of a system failure, while the UNDO segment provides rollback for transactions and read consistency.

Blocks of UNDO segments are cached in the System Global Area just like blocks of tables and indexes. If UNDO segment blocks are unused for a period of time, they may be aged out of the cache and written to disk.

UNDO segments are also used to implement Flashback features, which are described in Chapter 4 of this book.

System Change Number (SCN)
: To preserve the integrity of the data in the database and enforce any type of serialization, it is critical to keep track of the order in which actions were performed. Oracle uses the System Change Number as an absolute determinant of the order of transactions.

  The SCN is a logical timestamp that tracks the order in which transactions begin. Oracle uses the SCN information in the redo log to reproduce transactions in the original and correct order when applying redo. Oracle also uses the SCN to determine when to clean up information in rollback segments that are no longer needed, as you will see in the following sections.

Since Oracle Database 10*g*, there is a pseudocolumn on each row that contains the SCN, ORA_ROWSCN. You can use this to quickly determine if a row has been updated since it was retrieved by comparing the value read from this pseudocolumn at the start of a transaction with the value read from this pseudocolumn at the end of the transaction.

Locks in data blocks
: A database must have a way of determining if a particular row is locked. Most databases keep a list of locks in memory, which are managed by a lock manager process. Oracle keeps locks with an area of the actual block in which the row is stored. A data block is the smallest amount of data that can be read from disk for an Oracle Database, so whenever the row is requested, the block is read, and the lock is available within the block. Although the lock indicators are kept within a block, each lock affects only an individual row within the block.

In addition to the above features, which directly pertain to multiversion read consistency, another implementation feature in Oracle provides a greater level of concurrency in large user populations:

Nonescalating row locks
: To reduce the overhead of the lock-management process, other databases will sometimes *escalate* locks to a higher level of granularity within the database. For example, if a certain percentage of rows in a table are locked, the database will escalate the lock to a table lock, which locks all the rows in a table, including rows that aren't specifically used by the SQL statement in question. Although lock escalation reduces the number of locks the lock manager process has to handle, this escalation causes unaffected rows to be locked. With Oracle, the lock indicator is stored within the data block itself, so there is no increase in overhead for a lock manager when the number of locks increases. Consequently, there is never any need for Oracle to escalate a lock.

A lock manager called the Distributed Lock Manager (DLM) has historically been used with Oracle Parallel Server to track locks across multiple instances of Oracle. This is a completely different and separate locking scheme that doesn't affect the way Oracle handles row locks. The DLM technology used in Oracle Parallel Server was improved and integrated into a core product in Oracle9*i*, Real Application Clusters. Real Application Clusters are described in more detail in Chapter 9.

How Oracle Handles Locking

If you've read this chapter from the beginning, you should now know enough about the concepts of concurrency and the features of Oracle to understand how the Oracle Database handles multiuser access. However, to make it perfectly clear how these features interact, we'll walk you through three scenarios: a simple write to the database, a situation in which two users attempt to write to the same row in the same table, and a read that takes place in the midst of conflicting updates.

For the purposes of these examples, we'll use the scenario of one or two users modifying the EMP table, a part of the standard sample Oracle schema that lists data about employees via a form.

A Simple Write Operation

This example describes a simple write operation, in which one user is writing to a row in the database. In this example, an HR clerk wants to update the name for an employee. Assume that the HR clerk already has the employee record on-screen. The steps from this point are as follows:

1. The client modifies the employee name on the screen. The client process sends a SQL UPDATE statement over the network to the server process.
2. The server process obtains a System Change Number and reads the data block containing the target row.
3. The server records row lock information in the data block.
4. The server writes the old image of the data to the redo buffers in memory, and then writes the changes to an UNDO segment and modifies the employee data, which includes writing the SCN to the ORA_ROWSCN pseudocolumn in Oracle Database 10*g* or newer database releases.
5. The server process writes the redo buffers to disk, and then writes the UNDO segments and the changed data to disk. The UNDO segment changes are part of the redo, since the redo log stores all changes coming from the transaction.
6. The HR clerk commits the transaction.

7. Log Writer (LGWR) writes the redo information for the entire transaction, including the SCN that marks the time the transaction was committed, from the redo log buffer to the current redo logfile on disk. When the operating system confirms that the write to the redo logfile has successfully completed, the transaction is considered committed.
8. The server process sends a message to the client confirming the commit.

Oracle Database 10*g* Release 2 introduced the ability to have the server process return control to the client without waiting for all the redo information to be written. The plus side of this enhancement is that high-volume OLTP applications may benefit from improved performance. The downside of this feature is that it opens a window of vulnerability—the database could crash after a transaction had been committed, but before the redo was written, which would make it impossible to recover the committed transaction, so this feature should be used with caution.

Oracle Database 12*c* introduces a new feature called Transaction Guard. If you followed the description above closely, you can see that a transaction could be between step 7 and step 8 when some type of failure occurs that prevents the message relaying the successful commit to the application. (This failure could have nothing to do with the database and be quite short-lived—such as a network failure.)This series of events would mean that the Oracle Database has committed data, but the application does not know whether the commit failed or succeeded. Transaction Guard provides an API that allows an application to specifically check on the outcome of a potentially failed transaction. Transaction Guard is described in more detail in Chapter 9.

A Conflicting Write Operation

The write operation previously described is a little different if there are two users, Client A and Client B, who are trying to modify the same row of data at the same time. The steps are as follows:

1. Client A modifies the employee name on the screen. Client A sends a SQL UPDATE statement over the network to the server process.
2. The server process obtains an SCN for the statement and reads the data block containing the target row.
3. The server records row lock information in the data block.
4. The server process writes the changes to the redo log buffer.
5. The server process copies the old image of the employee data about to be changed to an UNDO segment. Once the server process has completed this work, the process modifies the employee data, which includes writing the SCN to the ORA_ROWSCN pseudocolumn in Oracle Database 10*g* or newer database releases.

6. Client B modifies the employee name on the screen and sends a SQL UPDATE statement to the server.
7. The server process obtains an SCN and reads the data block containing the target row.
8. The server process sees that there is a lock on the target row from the information in the header of the data block, so it takes one of two actions. If the isolation level on Client B's transaction is READ COMMITTED, the server process waits for the blocking transaction to complete. If the isolation level for Client B's transaction is SERIALIZABLE, an error is returned to the client.
9. Client A commits the transaction, the server process takes the appropriate action, and the server sends a message to Client A confirming the commit.
10. If Client B executed the SQL statement with the READ COMMITTED isolation level, the SQL statement then proceeds through its normal operation.

The previous example illustrates the default behavior of Oracle when it detects a problem caused by a potential lost update. Because the SERIALIZABLE isolation level has a more drastic effect when it detects a write conflict than the READ COMMITTED isolation level, many developers prefer the latter level. They can avoid some of the potential conflicts by either checking for changes prior to issuing an update (by comparing values in a row or using the Oracle Database 10*g* or later row SCN) or using the SELECT FOR UPDATE syntax in their SQL to avoid the problem altogether.

A Read Operation

You can really appreciate the beauty of Oracle's read consistency model by looking at the more common scenario of one user reading data and one user writing to the same row of data. In this scenario, Client A is reading a series of rows from the EMP table, while Client B modifies a row before it is read by Client A, but after Client A begins her transaction:

1. Client A sends a SQL SELECT statement over the network to the server process.
2. The server process obtains an SCN for the statement and begins to read the requested data for the query. For each data block that it reads, it compares the SCN of the SELECT statement with the SCNs for any transactions for the relevant rows of the data block. If the server finds a transaction with a later SCN than the current SELECT statement, the server process uses data in the UNDO segments to create a "consistent read" version of the data block, current as of the time the SELECT was issued. This is what provides the multiversion read consistency (MVRC) and avoids the need for Oracle to use read locks on data. If a row has been updated since the transaction started, Oracle simply gets the earlier version of the data for a consistent view.

3. Client B sends a SQL UPDATE statement for a row in the EMP table that has not yet been read by Client A's SELECT statement. The server process gets an SCN for the statement and begins the operation.
4. Client B commits his changes. The server process completes the operation, which includes recording information in the data block that contains the modified row that allows Oracle to determine the SCN for the update transaction.
5. The server process for Client A's read operation comes to the newly modified block. It sees that the data block contains changes made by a transaction that has an SCN that is later than the SCN of the SELECT statement. The server process looks in the data block header, which has a pointer to the UNDO segment that contains the data as it existed when Client A's transaction started. The UNDO segment uses the old version of the data to create a version of the block as it existed when the SELECT statement started. Client A's SELECT statement reads the desired rows from this consistent version of the data block.

Figure 8-2 illustrates the process of reading with multiversion read consistency.

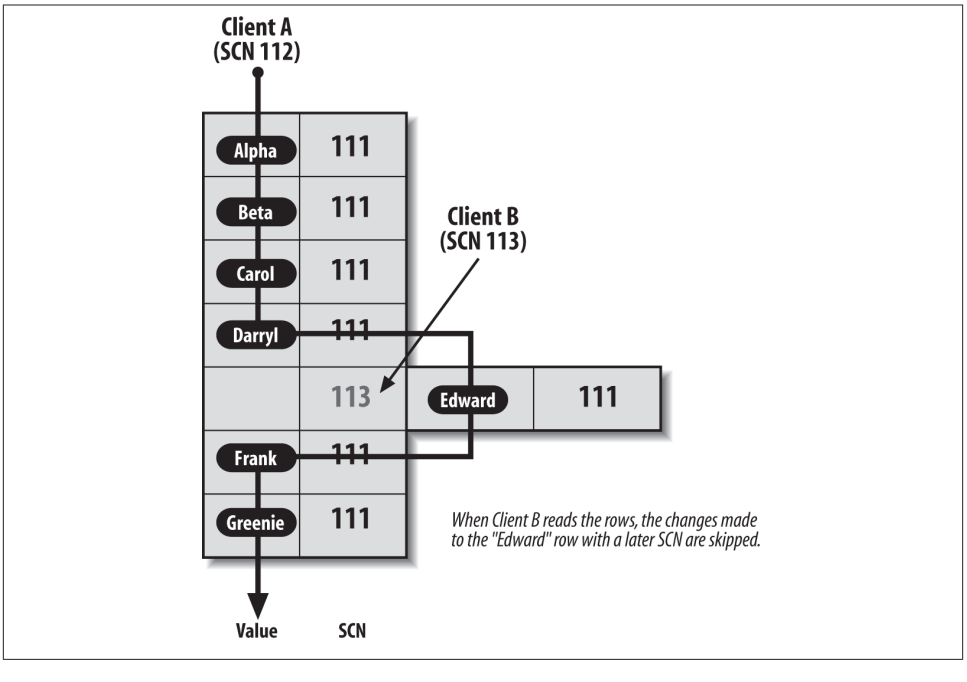


Figure 8-2. Multiversion read consistency

We explained how MVRC works with two users for the sake of clarity. But imagine a database supporting one or more enterprise applications, with thousands of simultaneous users. Oracle's concurrency handling could avoid an enormous amount of

contention and performance degradation in a heavy use scenario—in fact, the greater the workload, the greater the benefits of MVRC.

Concurrent Access and Performance

When you read through all the steps involved in the above processes, you might think that Oracle would be a very slow database. This is not at all true. Oracle has consistently turned in benchmarks that show it to be one of the fastest databases, if not the fastest, on the market today.

Oracle provides good performance while implementing multiversion read consistency by minimizing and deferring unnecessary I/O operations. To assure the integrity of the data in a database, the database must be able to recover in the event of a system failure. This means that there must be a way to ensure that the data in the database accurately reflects the state of the committed data at the time of the crash. Oracle can do this by writing changed data to the database whenever a transaction commits. However, the redo log contains much less information than the entire data block for the changed data, so it's much "cheaper" to write to disk. Oracle writes the redo information to disk as soon as a transaction commits and defers writing the changed data blocks to the database until several sets of changed blocks can be written together. Oracle can restore the database using the redo logs, and these procedures cut down on time-consuming I/O operations.

However, when you're considering the performance of a database, you have to think about more than simple I/O operations. It doesn't really matter how fast your database runs if your transaction is waiting for another transaction to release a lock. A faster database may complete the blocking transaction faster, but your transaction is still at a dead stop until the blocking transaction completes.

Because most databases perform a mixture of reading and writing, and because Oracle is one of the only databases on the market that doesn't use read locks, Oracle will essentially always deliver the lowest amount of database contention. Less contention equals greater throughput for a mixed application load.

There is also more than one type of performance. Performance for database operations is measured in milliseconds of response time; performance for application developers is measured in months of development time. Because Oracle provides much less contention with its read consistency model, developers have to spend less time adding workarounds to their applications to handle the results of contention.

It's not as though Oracle is the only database to give you a concurrency solution you can use to implement applications that provide adequate data integrity. But the multiversion read consistency model makes it easy for you to get a consistent view of data without excessive contention and without having to write workarounds in your application. If it sounds as if we're big fans of Oracle's locking scheme, well—we are.

Workspaces

A *workspace* is a way to isolate data from changes in the general database environment. Workspace Manager accomplishes this by creating workspace-specific versions of data. When you create a workspace, you essentially create a snapshot of the data in the workspace at a specific point in time. Further changes to that data from outside the workspace do not affect the view of the data in the workspace, and changes made to data within the workspace are not seen by users outside the workspace. And changes to data within a workspace are visible only to other workspace users.

Workspaces allow you to essentially create separate data environments for specialized usage. You can capture data at a certain point in time for historical analysis and can also perform various types of "what if" analysis, testing to see how changes would affect the overall composition of the data without disturbing the main production database. Both of these options would normally require you to create a duplicate database, so workspaces can save you time and resources.

Workspace Implementation

The key to workspaces is the support of multiple versions of the same data. To use workspaces to version data in a table, you must first enable the table for versioning. Workspace Manager can version-enable one or more user tables in the database. The unit of versioning is a row. When a table is version-enabled, all rows in the table can support multiple versions of the data. Versioned rows are stored in the same table as the original rows. The versioning infrastructure is not visible to the users of the database, and application SQL statements to select, insert, modify, and delete data continue to work in the usual way with version-enabled tables. Workspace Manager version-enables a table by renaming the table, adding a few columns to the table to store versioning metadata, creating a view on the version-enabled table using the original table name, and defining INSTEAD OF triggers on the view for SQL DML operations.

The workspace keeps changes to the data only to minimize the size of the workspace data and avoid data duplication.

You can have a hierarchy of workspaces, and a workspace can have more than one parent. All workspace operations, described in the next sections, affect a workspace and its parent workspaces. Multiple levels of workspaces can give you finer granularity on the isolation of changes for workspace-enabled tables.

Oracle implements workspaces by adding metadata to the rows of a table. This metadata can include a timestamp as to when a change was made, which can help in analysis of workspace activity. This option works with savepoints to provide a history of changes made to each row version created by a savepoint. The timestamp allows users in a workspace to go back to any point in time and view the database from the perspective

of changes made in that workspace up to another point in time. You can think of this as a type of Flashback (described in Chapter 3) for a limited set of tables.

In addition, you can specify that a particular version of data in a workspace is valid only for a specific time period. For instance, you could make a change to data that would be visible to workspace users for the next 24 hours and that would then disappear. This type of functionality has been added to the core database in Oracle Database 12*c* with the valid time temporal feature, described in Chapter 4.

Workspaces have their own locking mechanisms that apply only to other workspace users. You can exclusively lock a row of data in a workspace, but this lock prevents access only to that row for other workspace users. The underlying data could still be accessed or changed by users who are not part of the workspace. This additional locking makes sense, since both locks and workspaces are meant to isolate data from changes. A workspace exists outside the boundaries of the standard database, so workspace locks and standard database locks do not directly interact.

Workspace Operations

There are three basic operations that apply to workspaces:

Rollback
: You can roll back changes to a workspace to return the workspace to the point in time when the workspace was created. You can also designate savepoints, which allow you to roll back the changes in a workspace to a subsequent point in time.

Refresh
: Refreshing a workspace means bringing the data in a workspace into agreement with the same data in the overall database. This capability could be used if you chose to create a workspace with a snapshot of the data at the end of a day. At midnight, you would refresh the workspace to make the workspace reflect the data from the previous day.

Merge
: A merge operation rolls changes made in a workspace into its parent workspace.

As you can imagine, both the refresh and the merge operations could end up with conflicts between data values in the workspace and its parent. Workspace management keeps track of conflicts on a per-table basis; you can resolve the conflicts manually.

Workspace Enhancements

Workspace Manager is tightly integrated with the Oracle Database. Oracle Database 10*g* Workspace Manager enhancements included the ability to export and import version-enabled tables, to use SQL*Loader to bulk load data into version-enabled tables,

to trigger events based on workspace operations, and to define workspaces that are continually refreshed.

Oracle Database 11*g* continues the stream of enhancements to workspaces, providing support for optimizer hints and more data maintenance operations on workspace-enabled tables. Oracle Database 12*c* includes enhancements to Workspace Manager and performance enhancements for the query optimizer to create better execution plans for workspace queries.

CHAPTER 9

Oracle and Transaction Processing

The value of information systems is clear from the ever-increasing number of transactions processed by the world's databases. Transactions form the foundation of business computing systems. In fact, transaction processing (TP) was the impetus for business computing as we know it today. The batch-oriented automation of core business processes like accounting and payroll drove the progress in mainframe computing through the 1970s and 1980s. Along the way, TP began the shift from batch to users interacting directly with systems, and online transaction processing (OLTP) was born. In the 1980s, the computing infrastructure shifted from large centralized mainframes with dumb terminals to decentralized client/server computing with graphical user interfaces (GUIs) running on PCs and accessing databases on other machines over a network.

The client/server revolution provided a much better user interface and reduced the cost of hardware and software, but it also introduced additional complexity in systems development, management, and deployment. After a decade of use, system administrators were being slowly overwhelmed by the task of managing thousands of client machines and dozens of servers, so the latter half of the 1990s saw a return to centralization, including the grid (introduced in Chapter 1). Throughout all of these shifts, Oracle Databases have continued to use their inherent architecture and constant enhancements to service the ever-growing load of transactions.

This chapter looks at the features of the Oracle Database that contribute to its ability to handle large transaction loads. Although many of the specific features covered in this chapter are touched upon in other chapters of this book, this chapter examines all of these features in light of their use in large OLTP systems.

OLTP Basics

Before we discuss how Oracle specifically handles OLTP, we'll start by presenting a common definition of online transaction processing.

What Is a Transaction?

The concept of a transaction—and the relevant Oracle mechanics for dealing with the integrity of transactions—was discussed in Chapter 8. To recap that discussion, a *transaction* is a logical unit of work that must succeed or fail in its entirety. Each transaction typically involves one or more Data Manipulation Language (DML) statements such as INSERT, UPDATE, or DELETE, and ends with either a COMMIT to make the changes permanent or a ROLLBACK to undo the changes.

The industry bible for OLTP, *Transaction Processing: Concepts and Techniques*, by Jim Gray and Andreas Reuter (Morgan Kaufmann; see Appendix Additional Resources), introduced the notion of the *ACID* properties of a transaction. A transaction must be the following:

Atomic
: The entire transaction succeeds or fails as a complete unit.

Consistent
: A completed transaction leaves the affected data in a consistent or correct state.

Isolated
: Each transaction executes in isolation and doesn't affect the states of others.

Durable
: The changes resulting from committed transactions are persistent.

If transactions execute serially—one after the other—their use of ACID properties can be relatively easily guaranteed. Each transaction starts with the consistent state of the previous transaction and, in turn, leaves a consistent state for the next transaction. Concurrent usage introduces the need for sophisticated locking and other coordination mechanisms to preserve the ACID properties of concurrent transactions while delivering throughput and performance. Chapter 8 covered Oracle's handling of locking and concurrency in depth.

What Does OLTP Mean?

Online transaction processing can be defined in different ways: as a type of computing with certain characteristics, or as a type of computing in contrast to more traditional batch processing.

General characteristics

Most OLTP systems share some of the following general characteristics:

High transaction volumes and large user populations
: OLTP systems are the key operational systems for many companies, so these systems typically support the highest volume and largest communities of any systems in the organization.

Well-defined performance requirements
: OLTP systems are central to core business operations, so users must be able to depend on a consistent response time. OLTP systems often involve service level agreements that state the expected response times.

High availability
: These systems are typically deemed mission-critical with significant costs resulting from downtime.

Scalability
: The ability to increase transaction volumes without significant degradation in performance allows OLTP systems to handle fluctuations in business activity.

In short, OLTP systems must be able to deliver consistent performance at any time, regardless of system load. Anything that affects these core systems can produce a ripple effect throughout your entire organization, affecting both revenue and profitability.

Online and batch transaction processing

Online transaction processing implies direct and conversational interaction between the transaction processing system and its users. Users enter and query data using forms that interact with the backend database. Editing and validation of data occur at the time the transactions are submitted by users.

Batch processing occurs without user interaction. Batches of transactions are fed from source files to the operational system. Errors are typically reported in exception files or logs and are reviewed by users or operators later on. Virtually all OLTP systems have a batch component: jobs that can execute in off-peak hours for reporting, payroll runs, posting of accounting entries, and so on.

Many large companies have batch-oriented mainframe systems that are so thoroughly embedded in the corporate infrastructure that they cannot be replaced or removed. A common practice is to "frontend" these legacy systems with OLTP systems that provide more modern interfaces. Users interact with the OLTP system to enter transactions. Batch files are extracted from the OLTP system and fed into the downstream legacy applications.

Once the batch processing is done, extracts are produced from the batch systems and are used to refresh the OLTP systems. This extraction process provides the users with a more sophisticated interface with online validation and editing, but it preserves the flow of data through the entrenched batch systems. While this process seems costly, it's typically more attractive than the major surgery that would replace older systems. To compound the difficulty, in some cases the documentation of these older systems is incomplete and the employees who understand the inner workings have retired or moved on.

The financial services industry is a leader in information technology for transaction processing, so this notion of feeding legacy downstream applications is very common in banks and insurance companies. For example, users often enter insurance claims into frontend online systems. Once all the data has been entered, if the claim has been approved, it's extracted and fed into legacy systems for further processing and payment.

Oracle features, such as transportable tablespaces (discussed in Chapter 13), are aimed in part at providing the functionality required by distributed OLTP systems in a more timely fashion than traditional batch jobs.

OLTP Versus Business Intelligence

Mixed workloads—OLTP and reporting—are the source of many performance challenges and the topic of intense debate. The data warehousing industry had its genesis in the realization that OLTP systems could not realistically provide the needed transaction throughput while supporting the enormous amount of historical data and ad hoc query workload that business analysts needed for things like multiyear trend analysis.

The issue isn't simply one of adequate machine horsepower; rather, it's the way data is modeled, stored, and accessed, which is typically quite different. In OLTP, the design centers on analyzing and automating business processes to provide consistent performance for a well-known set of transactions and users. The workload revolves around large numbers of short and well-defined transactions—with a fairly significant percentage of write transactions.

Business intelligence typically operates on larger data stores that frequently are assembled from multiple data sources and contain long histories. The schema design for data warehouses is usually very different from the fully normalized design best suited for OLTP data stores. And data warehouses can support ad hoc queries that, because of their complexity and the amount of data accessed, can place significant loads on a system with only a handful of requests.

Reporting and query functions are part of an OLTP system, but the scope and frequency are typically more controlled than in a data warehouse environment. For example, a banking OLTP system will include queries for customer status and account balances, but potentially not multiyear transaction patterns.

An OLTP system typically provides forms that allow well-targeted queries that are executed efficiently and don't consume undue resources. However, hard and fast rules—for example, that OLTP systems don't include extensive query facilities—don't necessarily hold true. The I/O performed by most OLTP systems tends to be approximately 70–80 percent reads and 20–30 percent writes. Most transactions involve the querying of data, such as product codes, customer names, account balances, inventory levels, and so on. Users submitting tuned queries for specific business functions are a key part of OLTP. Ad hoc queries across broad data sets are not.

Business intelligence data warehousing systems and OLTP systems could access much of the same data, but these types of systems also typically have different requirements in terms of CPU, memory, and data layout, which makes supporting a mixed workload less than optimal for both types of processing. Real Application Clusters, with dynamic service provisioning since Oracle Database 10*g*, makes it possible to allocate individual nodes for individual workloads. It also makes it more feasible to deploy these mixed workloads to a single database (albeit with multiple database instances).

Transactions and High Availability

As discussed in the previous chapter, the foundation of the transaction is the commit. Once a transaction is committed, it is a part of the core data store—if the transaction is not committed, its data is not visible to other users. Logically, this description is completely accurate. In operational terms, there is an additional step that is part of this process. Because commit operations are requested by a client application, the commit first takes place in the database and is then acknowledged to the requesting client. A failure could occur between the database commit operation and the reception of the acknowledgment. When this occurs, the transaction is in a grey area, with a crucial implication. If, when the system recovers, a transaction was committed on the database, it should not be redone; if the transaction did not complete, it should be redone. All the client knows is that the acknowledgment was not received. In a high-volume, mission-critical system, many crucial transactions may be in this state, even due to very temporary problems like network outages.

Oracle Database 12*c* introduces a new concept that addresses this specific problem, called *Transaction Guard*. Transaction Guard tracks the state of transactions in the database and provides an API that allows developers to query this state. With this advance, an application can find out the true state of a transaction that did not complete and take appropriate actions.

Another feature, Application Continuity, is also part of Oracle Database 12*c*. Application Continuity allows developers to create *Database Requests* that can use the information provided by Transaction Guard to determine the state of a failed transaction and automatically take action to either redo the transaction or ignore the failure. This capability provides a higher level of consistency in the Oracle Database without forcing developers to create entire subsystems to track and correct incomplete transactions on their own.

Oracle's OLTP Heritage

Oracle has enjoyed tremendous growth as the database of choice for OLTP in the mid range computing environment. Oracle6 introduced nonescalating row-level locking and read consistency (two of the most important of Oracle's core OLTP features), but Oracle7 was really the enabler for Oracle's growth in OLTP. Oracle7 introduced many key features, including the following:

- Multi-Threaded Server (MTS), now known as shared server
- Shared SQL
- Stored procedures and triggers
- XA support
- Distributed transactions and two-phase commits
- Data replication
- Oracle Parallel Server (OPS)[1]

Oracle8 enhanced existing functionality and introduced additional OLTP-related features, including the following:

- Connection pooling
- Connection multiplexing
- Data partitioning
- Advanced Queuing (AQ)
- Index organized tables
- Internalization of the Distributed Lock Manager (DLM) for Oracle Parallel Server
- Internalization of the triggers for replicated tables and parallel propagation of replicated transactions

Oracle8*i* provided the following additional enhancements and technologies for OLTP:

- Support for Java internally in the database kernel
- Support for distributed component technologies: CORBA v2.0 and Enterprise JavaBeans (EJB) v1.0
- Publish/subscribe messaging based on Advanced Queuing
- Online index rebuild and reorganization
- Database Resource Manager (DBRM)
- Use of a standby database for queries
- Internalization of the replication packages used to apply transactions at the remote sites

Oracle9*i* continued this trend, with the introduction of Real Application Clusters, which extended the benefits of Oracle Parallel Server to OLTP applications. Since Oracle

1. OPS was actually available for DEC VMS in 1989 and for NCR Unix with the last production release of Oracle6 (version 6.0.36), but it became widely available, more stable, and more popular in Oracle7.

Database 10*g*, the capabilities of Real Application Clusters support deployment to a new computing model: grid computing. But many of the capabilities that enable OLTP with Oracle have been core to the database product for many years.

As described above, Oracle Database 12*c* extends the robustness of OLTP with Transaction Guard and Application Continuity, which practically eliminate the window of indeterminate transaction outcomes.

The remainder of this chapter examines many of these features in more depth.

Architectures for OLTP

Although all OLTP systems are oriented toward the same goals, there are several different underlying system architectures that you can use for the deployment of OLTP, including the traditional two-tier model, a three-tier model, and a centralized model that encompasses the use of the Web and the grid.

Traditional Two-Tier Client/Server

The late 1980s saw the rise of two-tier client/server applications. In this configuration, PCs acted as clients accessing a separate database server over a network. The client ran both the GUI and the application logic, giving rise to the term *fat clients*. The database server processed SQL statements and returned the requested results back to the clients. While the database server component of these systems was relatively simple to develop and maintain using visual tools, client components were difficult to deploy and maintain —they required fairly high-bandwidth networks and the installation and regular upgrading of specific client software on every user's PC.

Figure 9-1 illustrates the two-tier architecture.

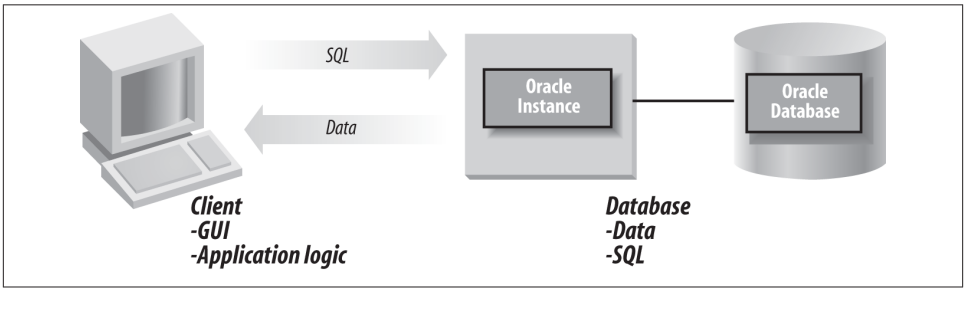


Figure 9-1. Two-tier client/server architecture

Stored Procedures

Oracle7 introduced stored procedures written in PL/SQL, Oracle's proprietary language for creating application logic. These procedures are stored in the database and executed by clients issuing remote procedure calls (RPCs) as opposed to executing SQL statements. Instead of issuing multiple SQL calls, occasionally with intermediate logic to accomplish a task, the client issues one procedure call, passing in the required parameters. The database executes all the required SQL and logic using the parameters it receives.

Stored procedures can also shield the client logic from internal changes to the data structures or program logic. As long as the parameters the client passed in and received back don't change, no changes are required in the client software. Stored procedures move a portion of the application logic from the client to the database server. By doing so, stored procedures can reduce the network traffic considerably. This capability increases the scalability of two-tier systems. Figure 9-2 illustrates a two-tier system with stored procedures.

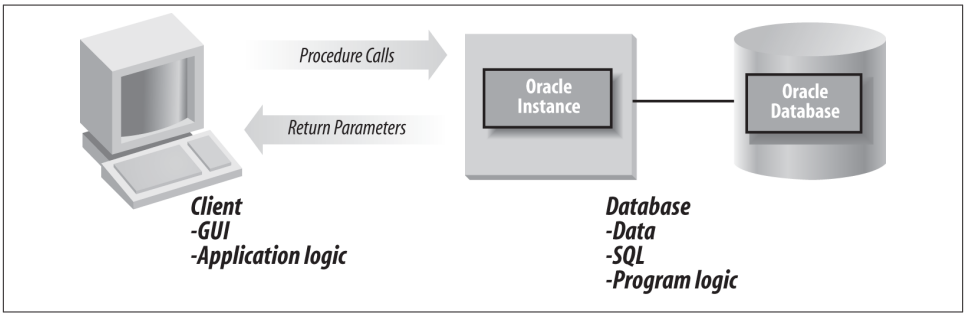


Figure 9-2. Two-tier system with stored procedures

Three-Tier Systems

The OLTP systems with the largest user populations and transaction throughput are typically deployed using a three-tier architecture. In the past, the three-tier architecture involved a transaction processing monitor, but now more frequently uses an application server. Clients access a transaction processing (TP) monitor or application server in the middle tier that, in turn, accesses a database server on the backend. The notion of a TP monitor dates back to the original mainframe OLTP systems. Of course, in the mainframe environment all logic ran on one machine. In an open system environment, application servers typically run on a separate machine (or machines), adding a middle tier between clients and the database server.

There are various classes of application servers:

- Older, proprietary servers such as Tuxedo from BEA Systems on Unix and Windows, or CICS from IBM on mainframes
- Industry-standard application servers based on Java 2 Enterprise Edition (J2EE)
- The Microsoft .NET application server environment as part of the Windows operating systems for servers, for example, Windows 2000 or Windows 2003

Application servers provide an environment for running services that clients call. The clients don't interact directly with the database server. Some examples of calling services provided by a TP monitor on a remote machine seem similar in many ways to the stored procedure architecture described in the previous section, which is why stored procedure-based systems are sometimes referred to as "TP-Lite."

Application servers provide additional valuable services, such as:

Funneling
: Like Oracle's shared servers, application servers leverage a pool of shared services across a larger user population. Instead of each user connecting directly to the database, the client calls a service running under the TP monitor or application server's control. The application servers invoke one of its services; the service interacts with the database. This architecture can be used to protect crucial data from direct client access.

Connection pooling
: The application server maintains a pool of shared, persistent database connections used to interact with the database on behalf of clients in handling their requests. This technique avoids the overhead of individual sessions for each client.

Load balancing
: Client requests are balanced across the multiple shared servers executing on one or more physical machines. The application servers can direct client service calls to the least-loaded server and can spawn additional shared servers as needed.

Fault tolerance
: The application server acts as a transaction manager; the monitor performs the commit or rollback of the transaction.[2] The underlying database becomes a resource manager, but doesn't control the transaction. If the database server fails while executing some transaction, the application server can resubmit the transaction to a surviving database server, as control of the transaction lies with the application server.

2. TP monitors usually control transactions using the X/Open Distributed Transaction Processing standard published by the X/Open standards body. A database that supports the XA interface can function as a resource manager under control of a TP monitor, which acts as a transaction manager.

This type of transaction resiliency is a hallmark of the older TP monitors such as Tuxedo, and the newer application servers and standards offer similar features.

Transaction routing
: The logic in the middle tier can direct transactions to specific database servers, increasing scalability.

Heterogeneous transactions
: Application servers can manage transactions across multiple heterogeneous database servers—for example, a transaction that updates data in Oracle and DB2.

While developing three-tier OLTP systems is complex and requires specialized skills, the benefits are substantial. Systems that use application servers provide higher scalability, availability, and flexibility than the simpler two-tier systems. Determining which architecture is appropriate for an OLTP system requires (among other things) careful evaluation and consideration of costs, available skills, workload profiles, scalability requirements, and availability requirements.

Figure 9-3 illustrates a three-tier system using an application server.

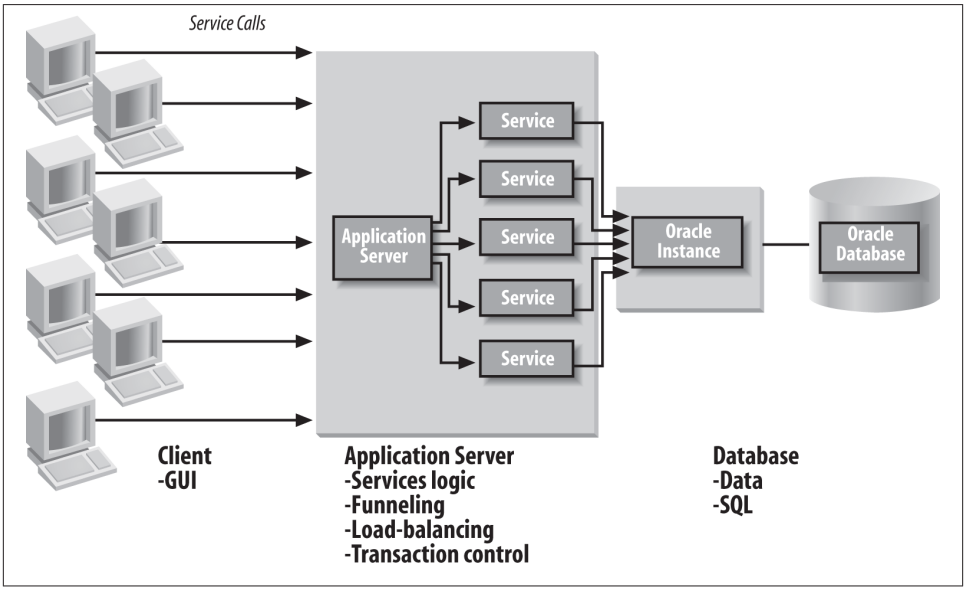


Figure 9-3. Three-tier architecture

Application Servers and Web Servers

The middle tier of web-based systems is usually an application server and/or a web server. These servers provide similar services to the application server previously described, but are more web-centric, dealing with HTTP, HTML, CGI, and Java.

J2EE and .NET application servers have evolved a great deal in the last decade and are the clear inheritors of the TP monitor legacy for today's *N*-tier systems. Different companies have different standards and preferences—the proprietary nature of .NET leads some firms to J2EE, while others prefer the tight integration of Microsoft's offerings. A detailed discussion of the relative merits of J2EE and .NET, and application server technology in general, is beyond the scope of this book. Suffice to say that application servers play an extremely important role in today's systems environment, and database management personnel need to understand *N*-tier systems architecture.

Figure 9-4 depicts an *N*-tier system with a client, web server, application server, and DBMS server.

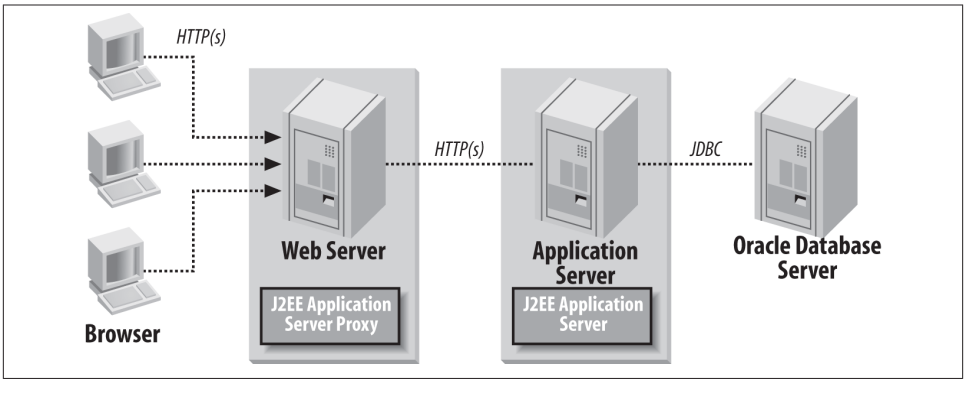


Figure 9-4. An N-tier system

The Grid

Oracle Database 10*g* introduced focus on another architecture variation—grid computing. The actual topology of the grid is not relevant to the discussion in this chapter, because the point of the grid is to provide an extremely simple user interface that transparently connects to a highly flexible source of computing power. In this way, the grid gives IT departments the ability to achieve the benefits of more complex architectures while not imposing undue complexity on users, and OLTP applications are deployed using grid computing resources.

In a similar fashion, the topology of underlying resources in the cloud are not visible to consumers, and the benefit of using complex architectures through a simple use model is one of the key features of cloud computing, which is the subject of Chapter 15.

OLTP and the Cloud

Use of centralized systems over the Internet, also known as *cloud computing*, is an increasingly popular approach. However, for truly high-volume OLTP systems, there are two significant obstacles to using the cloud.

The first is the unpredictable latency of the Internet. Not only can communications between a client application and database server take significantly longer over the Internet than in a dedicated system, but this latency can vary from one communication to the next, based on factors out of the control of the individual application, which is not an acceptable outcome when thousands of transactions are running simultaneously.

The second obstacle is based on the fact that all communications built on the HTTP protocol are, by definition, stateless. This means each interaction with a cloud database must be a complete transaction, since stateful communications are necessary for things like commits and rollbacks spanning over multiple interactions.

Many systems can run on the cloud despite these limitations, but high-end, mission-critical OLTP systems typically cannot accept these limitations.

Oracle Features for OLTP

Oracle has many features that contribute to OLTP performance, reliability, scalability, and availability. This section presents the basic attributes of many of these features. This section is by no means exhaustive; it is only intended to be an introduction. Please see the relevant Oracle documentation and third-party books for more information.

General Concurrency and Performance

As discussed in Chapter 8, Oracle has excellent support for concurrency and performance in OLTP systems. Some of the key features relevant to OLTP are as follows:

Nonescalating row-level locking
: Oracle locks only the rows a transaction works on and never escalates these locks to page-level or table-level locks. In some databases, which escalate row locks to page locks when enough rows have been locked on a page, contention can result from false lock contention when users want to work on unlocked rows but contend for locks that have escalated to higher granularity levels.

Multiversion read consistency
: Oracle provides statement-level and transaction-level data consistency without requiring read locks. A query is guaranteed to see only the data that was committed at the time the query started. The changes made by transactions that were in-flight but uncommitted at the time the query started won't be visible. Transactions that began after the query started and were committed before the query finishes also won't be seen by the query. Oracle uses UNDO segments to reproduce data as it existed at the time the query started. This capability avoids the unpleasant choice between allowing queries to see uncommitted data (known as dirty reads) or having readers block writers (and vice versa). It also provides a consistent snapshot view of data at a single point in time.

Shared SQL
: The parsing of a SQL statement is fairly CPU-intensive. Oracle caches parsed and optimized SQL statements in the shared SQL area within the shared pool. If another user executes a SQL statement that is cached, the parse and optimize overhead is avoided. The statements must be identical to be reused; no extra spaces, line feeds, or differences in capitalization are allowed. OLTP systems involve a large number of users executing the same application code. These systems provide an ideal opportunity for reusing shared SQL statements.

Stored outlines
: Oracle8*i* added support of execution plan stability, sometimes referred to as *bound plans*, with stored outlines. The route a SQL statement takes during execution is critical for high performance. Once application developers and DBAs have tuned a SQL statement for maximum efficiency, they can force the Oracle optimizer to use the same execution plan regardless of environmental changes. This provides critical stability and predictability in the face of software upgrades, schema changes, data-volume changes, and so on. Oracle9*i* added the capability for administrators to edit stored outlines.

  Since Oracle Database 10*g*, you can select better execution plans for the optimizer to use in conjunction with poorly written SQL to improve OLTP performance without having to rewrite the SQL. The SQL Tuning Advisor performs these advanced optimizations on SQL statements, and can then create an improved SQL profile for the statement. This profile is used instead of the original optimization plan at runtime.

Bind Variables and Shared SQL

As we've mentioned, Oracle's shared SQL is a key feature for building high-performance applications. In an OLTP application, similar SQL statements may be used repeatedly, but each SQL statement submitted will have different selection criteria contained in the WHERE clause to identify the different sets of rows on which to operate. Oracle can share SQL statements, but the statements must be absolutely identical.

To take advantage of this feature for statements that are identical except for specific values in a WHERE clause, you can use bind variables in your SQL statements. The values substituted for the bind variables in the SQL statement may be different, but the statement itself is the same.

Consider an example application for granting raises to employees. The application submits the following SQL:

```
UPDATE emp SET salary = salary * (1 + 0.1)
          WHERE empno = 123;
UPDATE emp SET salary = salary * (1 + 0.15)
          WHERE empno = 456;
```

These statements are clearly different; they update different employees identified by different employee numbers, and the employees receive different salary increases. To obtain the benefits of shared SQL, you can write the application to use bind variables for the percentage salary increase and the employee numbers, such as:

```
UPDATE emp SET salary = salary * (1 + :v_incr)
                 WHERE empno = :v_empno;
UPDATE emp SET salary = salary * (1 + :v_incr)
                 WHERE empno = :v_empno;
```

These statements are recognized as identical and would therefore be shared. The application would submit different values for the two variables, `:v_incr` and `:v_empno`—a percentage increase of 0.1 for employee 123 and 0.15 for employee 456. Oracle substitutes these actual values for the variables in the SQL. The substitution occurs during the phase of processing known as the *bind phase*, which follows the *parse phase* and *optimize phase*. For more details, see the relevant Oracle guide for your development language.

Oracle Database 10*g* and more recent versions include tuning tools that can easily spot this type of potential application optimization.

Scalability

Both the shared server and Database Resource Manager help Oracle support larger or mixed user populations.

Multi-Threaded Server/shared server

Oracle7 introduced the Multi-Threaded Server (MTS, renamed the shared server in Oracle9*i*) (described in Chapter 2) to allow Oracle to support larger user populations. While shared server / MTS reduced the number of server processes, each client still used its own physical network connection. The resources for network connections aren't unlimited, so Oracle8 introduced two solutions for increasing the capabilities of the actual network socket layer at the operating system level:

Oracle Net connection pooling
: Allows the client population to share a pool of shared physical network connections. Idle clients transparently "time out," and their network connections are returned to the pool to be used by active clients. Each idle client maintains a virtual connection with Oracle and will get another physical connection when activity resumes. With the Oracle security model, authentication is separate from a specific connection, so a single pooled connection can represent different users at different times. Connection pooling is suitable for applications with clients that connect but aren't highly active all the time.

Oracle Net Connection Manager
: Reduces the number of network connections used on the database server. Clients connect to a middle-tier machine running the Oracle Net Connection Manager (CMAN). The Connection Manager multiplexes the traffic for multiple clients into one network connection per Oracle Net dispatcher on the database server. Unlike connection pooling, there is no notion of "timeout" for a client's virtual network connection. The Oracle network topology can include multiple machines running the Connection Manager to provide additional scalability and fault tolerance.

In terms of scalability, you can think of connection pooling as the middleweight solution and multiplexing via Connection Manager as the heavyweight solution. Figure 9-5 illustrates these two network scaling technologies.

Connection Manager became more flexible in Oracle Database 10*g*, with the added ability to dynamically alter configuration parameters without shutting down Connection Manager and improved access rules to filter CMAN traffic.

Oracle Database 11*g* added Database Resident Connection Pooling, which can be even more efficient for high-volume, middle-tier connections to the database. Database Resident Connection Pooling eliminates the need to store session information by pooling sessions as well as servers, and this feature also eliminates the need to use a dispatcher in communications between the middle tier and the database.

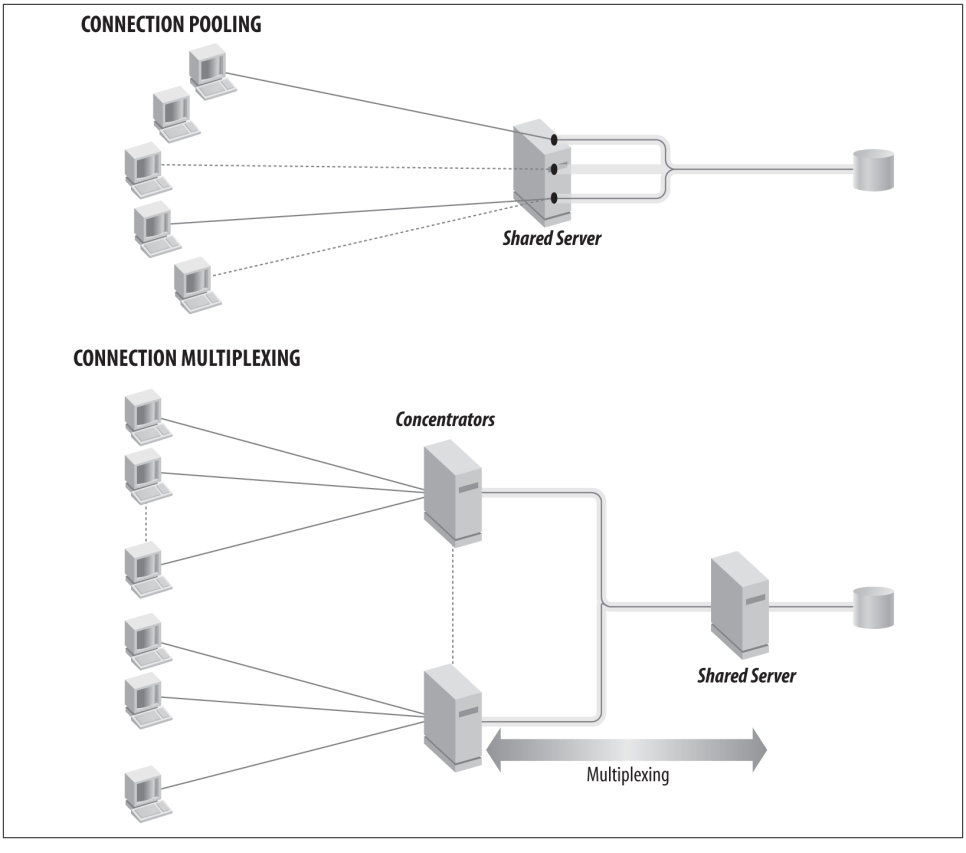


Figure 9-5. Network scaling in Oracle Net

Database Resource Manager

Oracle8*i* introduced the Database Resource Manager (DRM) to simplify and automate the management of mixed workloads in which different users access the same database for different purposes. You can define different consumer groups to contain different groups of users. The DRM allocates CPU and parallelism resources to consumer groups based on resource plans. A resource plan defines limits for the amount of a particular computer resource a group of users can use. This allows the DBA to ensure that certain types of users receive sufficient machine resources to meet performance requirements. You can also use a set of rules to move a particular user from one consumer group to another, such as forcing a user into a lower priority consumer group after they have used a certain amount of CPU time.

For example, you can allocate 80 percent of the CPU resources to order-entry users, with the remaining 20 percent allocated to users asking for reports. This allocation prevents reporting users from dominating the machine while order-entry users are

working. If the order-entry users aren't using all the allocated resources, the reporting users can use more than their allotted percentage. If the order-entry workload increases, the reporting users will be cut back to respect their 20 percent allocation. In other words, the order-entry users will get up to 80 percent of CPU time, as needed, while the users asking for reports will get at least 20 percent of the CPU time, and more depending on how much the order-entry group is using. With the DRM, you can dynamically alter the details of the plan without shutting down the instance. Since Oracle Database 11*g* R2, you can define an upper limit of CPU resources that is always enforced; prior to this enhancement, limits were only enforced when a CPU was oversubscribed.

Oracle9*i* added a number of significant improvements to the Database Resource Manager. The DRM now allows a DBA to specify the number of active sessions available to a consumer group. Any additional connection requests for the consumer group are queued. By limiting the number of active connections, you can start to avoid the situation where a request comes in that pushes the resource requirements for a group over the limit and affects all the other users in that group.

Oracle9*i* also added to the Database Resource Manager the ability to proactively estimate the amount of CPU that an operation will require. If an operation looks as if it will exceed the maximum CPU time specified for a resource group, the operation will not be executed, which can prevent inappropriately large operations from even starting.

Since Oracle9*i*, the DRM can also automatically switch a consumer group to another consumer group if that group is active for too long. This feature could be used to automatically switch a consumer group oriented toward short OLTP operations to another group that would be more appropriate for batch operations.

Since Oracle Database 10*g*, you can define a consumer group by the service name, application, host machine, or operating system username of a user.

Database Resource Manager was extended in connection with the release of Exadata. On this engineered system, resource plans can include the same type of distribution of I/O bandwidth between database instances and storage nodes that is available for CPU. Oracle Database 12*c* brings the Oracle Multitenant option and pluggable databases into the picture for Database Resource Manager.

Real Application Clusters

Arguably, the biggest advance in Oracle9*i* was a feature called Real Application Clusters. Real Application Clusters (RAC) was a new version of technology replacing Oracle Parallel Server (OPS).

In the first edition of this book, we described OPS as a feature that could be used for improving performance and scalability for certain data warehouse-style applications—applications in which data could be partitioned in logical ways and applications that

primarily supported read activity. The reason why use of OPS was mostly limited to data warehousing implementations was the phenomenon known as *pinging*.

In the world of both OPS and RAC, multiple machines access the same database files on shared disk (either physically attached or appearing as physically attached through software), as shown in Figure 9-6.

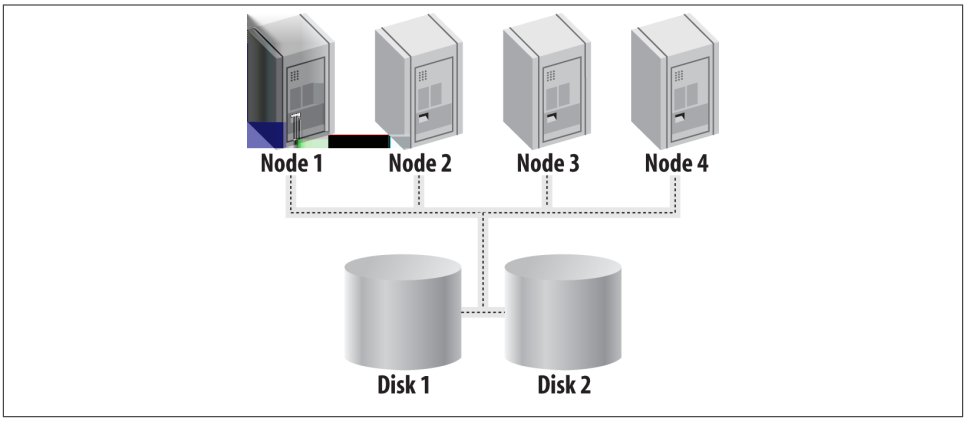


Figure 9-6. RAC architecture

This architecture allows you to add more machines to a cluster of machines, which in turn adds more overall horsepower to the system. But there was a problem with the implementation of this architecture for OPS, stemming from the fact that a page can contain more than a single row. If one machine in a cluster wanted to modify a row in a page that was already being modified by another machine, that page had to be flushed to the database file on the shared disk—a scenario that was termed a *ping*. This chain of events caused extra disk I/O, which in turn decreased the overall performance of the solution based on the number of writes.

The traditional way around this problem was simply to avoid it—to use OPS only when a database would not cause pings with a lot of write operations, or to segregate writes so that they would not require data in use on another node. This limitation required you to carefully consider the type of application to which you would deploy OPS and sometimes forced you to actually modify the design of your application to work around OPS's limitations.

With Real Application Clusters, the problem caused by pings was eliminated. RAC fully supports the technology known as Cache Fusion. Cache Fusion makes all the data in every cache on every machine in a Real Application Cluster available to every other machine in the cluster. If one machine needs a block that is either being used by another machine or simply residing in the cache of another machine, the block is directly shipped to the requesting machine, usually over a very high-speed interconnect.

Cache Fusion means that you do not have to work around the problems of pinging. With Real Application Clusters, you will be able to see significant scalability improvements for most all applications, without any modifications. With that said, for OLTP applications deployed to RAC (where there are frequent modifications to indexes within a small set of leaf blocks), reverse key indexes might be used to distribute inserts across leaf keys in the index and eliminate possible performance issues for this special situation (see Chapter 4 for an explanation of reverse key indexes).

Real Application Clusters also delivers all the availability advantages that were a part of OPS. Because all the machines in a Real Application Cluster share the same disk, the failure of a single machine does not mean that the database as a whole has failed. The users connected to the failed machine have to be failed over to another machine in the cluster, but the database server itself will continue to operate.

As of Oracle Database 10*g*, the model implemented with RAC was extended beyond clusters to grid computing. Oracle now offers all the components you need to use to implement clusters on several operating system platforms as part of the Oracle software stack, including a volume manager and clusterware. In Oracle 10*g* Release 2, Oracle made it possible to monitor the different nodes in a cluster and to issue advisories to ensure better load balancing across the nodes.

Oracle Database 11*g* Release 2 added the concept of *server pools*, which provide a higher level of organization within a RAC cluster. A server pool can have multiple nodes within it, and you can designate an application to run on all instances within the pool or a single instance. Instances can be added to a pool based on defined policies to handle shifting workload requirements.

Exadata and OLTP

The Oracle Exadata Database Machine and the Exadata Storage Server software represented a significant enhancement to the overall Oracle Database story, delivering performance that was up to an order of magnitude better than any other platform for the Oracle Database. Exadata is discussed in more detail elsewhere in this book.

In the first release, Exadata, the Exadata Storage Server software, was primarily used to speed up data warehouse operations, since the most dramatic improvements in performance revolved around read operations for large numbers of rows, which are not that common in OLTP workloads.

With the release of the Exadata X3 models and corresponding Exadata Storage Server software in 2012, a new feature called Smart Flash Cache *writeback* was added. Writeback takes advantage of the speed of I/O to the Flash Cache in each Exadata Storage Cell. With writeback enabled, write operations to disk are sent to the Flash Cache instead of the much slower disk writes. This option eliminates the overhead of random disk

writes, providing more I/O bandwidth for other writes, such as redo writes, as well as providing fast access to the newly written data stored in the Flash Cache.

The latest version of Exadata marks data in the Flash Cache as dirty, so if there is a failure of the Flash Cache, those dirty rows can be recovered, protecting the all important integrity of the data. This recovery is typically very fast.

The improvements provided by the writeback feature help to deliver performance benefits to classic OLTP workloads as well as data warehouse workloads with Exadata.

High Availability

From an operational perspective, OLTP systems represent a company's electronic central nervous system, so the databases that support these systems must be highly available. Oracle has a number of features that contribute to high availability:

Standby database
: Oracle can provide database redundancy by maintaining a copy of the primary database on another machine, usually at another site. Redo logs from the primary server are shipped to the standby server and applied there to duplicate the production activity. Oracle8*i* introduced the automated shipping of redo logs to the standby site and the ability to open the standby database for read-only access for reporting.

  Oracle9*i* Release 2 introduced the concept of *logical standby*. With a logical standby database, the changes are propagated with SQL statements, rather than redo logs, which allow the logical standby database to be used for other database operations.

Transparent Application Failover (TAF)
: TAF is a programming interface that automatically connects a user session to another Oracle instance should the primary instance fail. Any queries that were in process are resumed from the point of the last row fetched for the result set.

GoldenGate
: GoldenGate is an Oracle Fusion Middleware offering that provides a method for asynchronous, or deferred, intersystem replication, allowing systems to operate more independently. Avoiding direct system dependencies can help to avoid "cascading" failures, allowing interconnected systems to continue to operate even if one system fails. For example, Golden Gate can enable change data capture among Oracle Databases and can be used with non-Oracle databases. These capabilities are described in more detail in Chapter 11.

Oracle Data Guard
: You can use Oracle Data Guard functionality to provide data redundancy. Changes made by transactions are replicated synchronously or asynchronously to other databases. If the primary database fails, the data is available from the other databases. Oracle Data Guard is described in more detail in Chapter 11.

Real Application Clusters
: Real Application Clusters can increase the scalability of the Oracle Database over multiple nodes in a cluster. But by supporting multiple instances with full access to the same database, RAC also provides the highest levels of availability for protection from the failure of a node in a clustered environment. If one node fails, the surviving nodes provide continued access to the database. Grid computing deployment further extends availability capabilities.

Oracle Database 11*g* provides a number of high-availability enhancements, including the ability to easily capture diagnostic information about database failures. For a more detailed discussion of high-availability features and options, see Chapter 11.

CHAPTER 10

Oracle Data Warehousing and Business Intelligence

A database is general purpose software and it serves as the basis of a platform that meets a variety of needs, including:

Recording and storing transactional data
: Reliably storing data and protecting each user's data from the effects of other users' changes

Retrieving data for ad hoc questions (queries) and historical reporting
: Providing a consistent and persistent view of the data

Analyzing data
: Summarizing and comparing data, detecting trends and data relationships, and forecasting

The last two capabilities are most often associated with a *data warehouse*, part of an infrastructure that provides *business intelligence* used in strategic and tactical business management of the corporation or organization. Such solutions expose valuable business information embedded in an organization's data stores—essentially creating additional value from existing data resources. Because of this, data warehousing is the focus of a great deal of interest from business constituencies.

Data warehousing and business intelligence solutions are widely deployed and continue to be a focus for further development in many organizations. There is a very simple reason behind this: such projects and solutions are seen as core to making business decisions, providing a return on investment that can be grasped by the business community.

This trend is not new. Oracle began adding data warehousing features to Oracle7 in the early 1990s. Ever since, additional features for warehousing and analytics appeared, enabling better performance, functionality, scalability, and management. In addition to the Oracle Database, Oracle offers tools for building and using a business intelligence infrastructure, including data movement and data transformation tools, business analysis tools and applications, and Big Data solutions.

A business intelligence infrastructure enables business analysts to determine:

- How a business scenario compares to past business results
- New business possibilities by looking at the data differently
- Possible future business outcomes
- How business actions can be changed to impact the future

This chapter introduces the basic concepts, technologies, and tools used in building data warehousing and business intelligence solutions. To help you understand how Oracle addresses infrastructure and analyzes issues, we'll first describe some of the basic terms and technologies.

Data Warehousing Basics

Why build a data warehouse? Why is the data in an online transaction processing (OLTP) database only part of a business intelligence solution? Where does Big Data fit in a data warehousing deployment strategy?

Data warehouse relational database platforms are often designed with the following characteristics in mind:

Strategic and tactical analyses can discern trends in data
: Data warehouses often are used in creation of reports based on aggregate values culled from enormous amounts of data. If OLTP databases were used to create such aggregates on the fly, the database resources used would impact the ability to process transactions in a timely manner. These ad hoc queries often take advantage of computer-intensive analytic functions embedded in the database. Furthermore, if data volumes of this size were entirely pushed to in-memory databases in a middle tier, the platform cost would be prohibitive.

A significant portion of the data in a data warehouse is often read-only, with infrequent updates
: Database manageability features can make it possible to deploy warehouses containing petabytes of data, even where near real-time updates of some of the data is occurring.

The data in source systems is not "clean" or consistent across systems

Data input to transactional systems, if not carefully controlled, is likely to contain errors and duplication. Often, a key portion of the data warehouse loading process involves elimination of these errors through data transformation. Since multiple source systems might differ in data definitions, data transformations during the ETL (extraction, transformation, and load) process can be used to modify data into a single common definition as well as improve its quality.

The design required for an efficient data warehouse differs from the standard normalized design for a relational database

Queries are typically submitted against a *fact table*, which may contain summarized data. The schema design often used, a *star schema*, lets you access facts quite flexibly along key *dimensions* or "lookup" values. (The star schema is described in more detail later in this chapter.) For instance, a business user may want to compare the total amount of sales, which comes from a fact table, by region, store in the region, and items, all of which can be considered key dimensions. Today's data warehouses often feature a *hybrid schema* that is a combination of the star schema common in previous-generation data marts with third normal form schema for detailed data that is common in OLTP systems and enterprise data warehouses.

The data warehouse often serves as a target for meaningful data found on Big Data platforms that optimally solve semi-structured data problems

Big Data can be described as semi-structured data containing data descriptors, data values, and other miscellaneous data bits produced by sensors, social media, and web-based data feeds. Given the amount of irrelevant data present, the processing goal on a Big Data platform is to map the data and reduce it to data of value (hence "MapReduce" callouts in programs written using languages such as Java and Python that refine this data). This subset of Big Data is usually fed to a data warehouse where it has value across the business and might be analyzed side by side with structured data.

The Evolution of Data Warehousing and Business Intelligence

Gathering business intelligence from data warehouses is not a new idea. The use of corporate data for strategic decision-making beyond simple tracking and day-to-day operations has been going on for almost as long as computing itself.

Quite early, builders and users of operational systems recognized potential business benefits of analyzing the data in complementary systems. In fact, much of the early growth in personal computers was tied to the use of spreadsheets that performed analyses using data downloaded from the operational systems. Business executives began to direct IT efforts toward building solutions to better understand the business using such data, leading to new business strategies. Today, solutions are commonly provided in business areas such as customer relationship management, sales and marketing

campaign analysis, product management and packaging, financial analysis, supply chain analysis, risk and fraud analysis, and a host of other areas.

In the 1980s, many organizations began using dedicated servers for these applications, collectively known then as *decision support systems* (DSS), supplementing their management information systems. Decision-support queries tended to be CPU, memory, and I/O intensive using read-only data. The characteristics of queries were much less predictable (e.g., more "ad hoc") than what had been experienced in OLTP systems. This led to the development of data stores for decision support apart from those for OLTP.

When Bill Inmon and others popularized the term "data warehouse" in the early 1990s, a formalized common infrastructure for building a solution came into being. The topology of business intelligence solutions continued to evolve, as the next section illustrates. Today's business intelligence solutions often include infrastructure that exposes data from relational data warehouses and also from specialty engines (e.g., OLAP, Big Data) and OLTP reporting systems.

A Topology for Business Intelligence

The classic data warehouse topology, serving as an enterprise-wide source of information, is represented by the multitier topology shown in Figure 10-1.

This topology developed over many years for a variety of reasons. Initial efforts at creating a single enterprise warehouse often resulted in "analysis paralysis." Just as efforts to define an enterprise-wide OLTP model can take years (due to cross-departmental politics and the scope of the effort), similar attempts in data warehousing also took much longer than business sponsors were willing to accept. These efforts were further hampered by the continually changing analysis requirements necessitated by a changing marketplace and the introduction of new datatypes.

Consequently, attempts at building such enterprise-wide models that would satisfy everyone often satisfied no one and left out critical data needed to run the business.

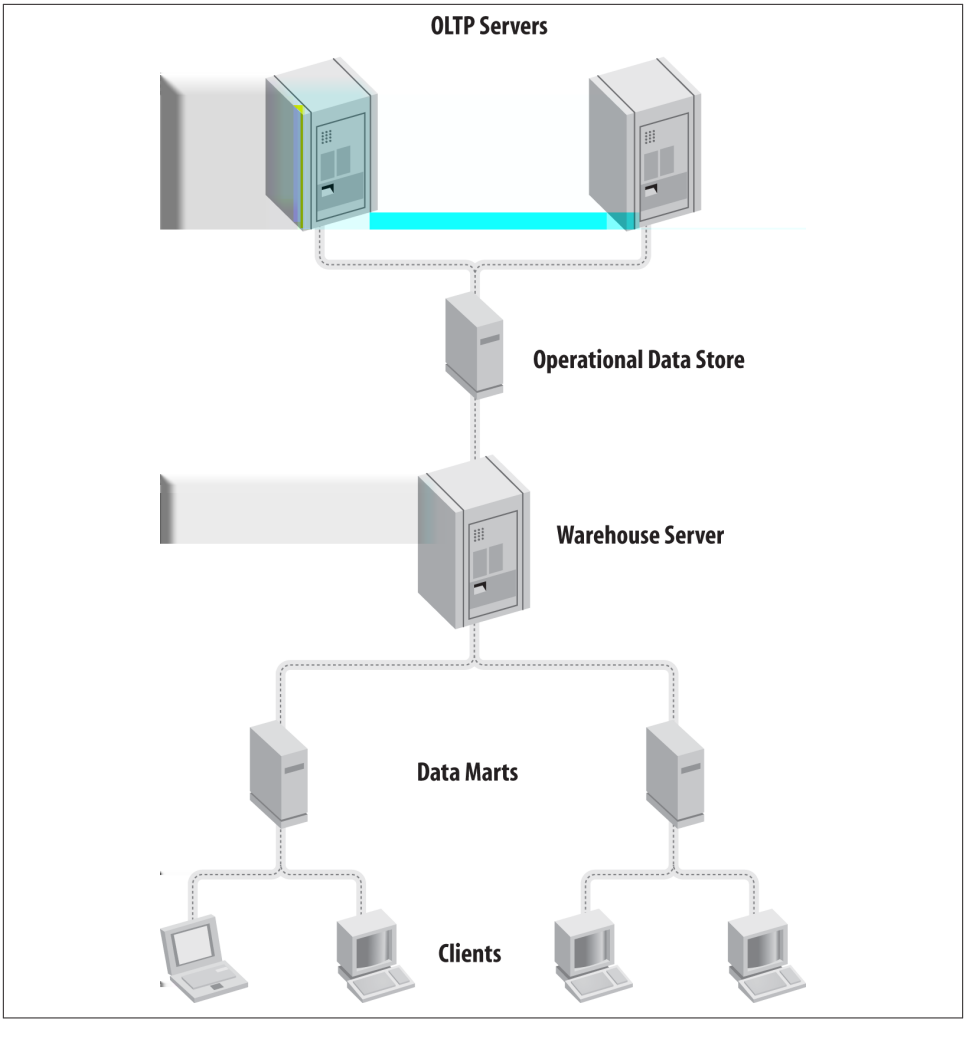


Figure 10-1. Typical initial business intelligence topology

Data Marts

When many early large-scale, enterprise-only data warehouse efforts ended in dismal failure, frustration and impatience followed. Some built department-focused independent *data marts* instead by extracting data from the appropriate operational source systems. Many data marts were initially quite successful because they fulfilled a specific business need relatively quickly.

However, problems began to surface. There was often no coordination between departments regarding basic definitions, such as "customer." If a senior manager asked the

same question of multiple departments, the answers provided by these independent data marts were often different, thus calling into question the validity of all of the marts. Many departments also encountered ongoing difficulty in managing these multiple data marts and in maintaining extractions from operational sources (which were often duplicated across multiple departments).

As architects took another look at their solutions, they began to realize that it was very important to have a consistent view of the detailed data at an enterprise data warehouse level. They also saw that data marts could solve business problems and provide return on investment in an incremental fashion. Today, most successful implementers simultaneously grow dependent data marts one business solution at a time while growing the enterprise warehouse server in an incremental fashion.

The currently accepted definition of a data mart is simply a subject- or application-specific data warehouse, usually implemented within a department. Typically, these data marts are built to solve specific business needs and may include a large number of summary tables. Data marts were initially thought of as being small, since not all the detail data for a department or data from other departments need be loaded in the mart. However, some marts get quite large as they incorporate data from outside sources (sometimes purchased) that isn't relevant in other parts of the business.

In some organizations, data marts are deployed to meet specific project goals with models optimized for performance for that particular project. Such data marts are retired when the project is completed and the hardware is reused for other projects. As the analysis requirements for a business change, the topology of any particular data warehouse is subject to evolution over time, so developers must be aware of this possibility.

Increasing focus on cost savings, manageability, and compliance are leading many to reexamine the wisdom of having a large number of physically separate data marts. As a result, consolidation of marts into the enterprise warehouse is a common trend. More recent versions of Oracle enable effective management of different user communities, helping to make such consolidation possible.

The Operational Data Store and Enterprise Warehouse

The *operational data store* (ODS) concept also grew in popularity in the 1990s. The ODS may best be described as a distribution center for current data. Like the OLTP servers, the schema is highly normalized and the data is recent. The ODS serves as a consolidation point for reporting and can give the business one location for viewing current data that crosses divisions or departments. The popularity of the ODS grew in part as a result of companies in the midst of acquisitions and mergers. These organizations often face mixed-application environments. The ODS can act as a staging location that can be used as the source for further transformations into a data warehouse or into data marts.

The warehouse server, or *enterprise data warehouse*, is a multisubject historical information store usually supporting multiple departments and often serving as the corporate database of record. When an ODS is established, the warehouse server often extracts data from the ODS. When an ODS isn't present, data for the warehouse is directly extracted and transformed from operational sources. External data may also feed the warehouse server.

As noted previously, platform consolidation is popular within these tiers today. The enterprise data warehouse can be the point of consolidation for the ODS and multiple data marts. Although different logical models remain, they are consolidated to a single platform and database.

OLTP Systems and Business Intelligence

True real-time data resides in the OLTP systems. Organizations can provide reporting from transaction processing systems side by side in portals or dashboards with information from data warehouse systems. A key to providing meaningful dashboards is to provide high-quality data with consistent meaning. The quality of data in OLTP systems is directly related to controlling data input to eliminate duplicate or error-prone entries.

Consistent meaning can be resolved using master data management (MDM) solutions. MDM solutions consist of data hubs that serve as a common reference point for data supporting key business measurements such as customers, products, or finance. Oracle offers a number of data hubs for these and other business areas to enable building out of such an infrastructure.

Projects that include data from data warehouses, OLTP systems, Big Data sources, and MDM solutions are called data integration projects. Most business intelligence deployments, at the time of publication of this edition, use just the data warehouse infrastructure as the primary source of historic data for business intelligence. The extraction, transformation, and loading (ETL) techniques applied to the data warehouse are designed to resolve differences in common data elements, to cleanse the data, and to provide a historical database of record.

Big Data and the Data Warehouse

Organizations are considering extending the business intelligence topology as they introduce Big Data platforms. These platforms are commonly defined as those that run a distribution of Apache Hadoop, an open source software framework ideal for analyzing unstructured or semi-structured data. At the base of any Hadoop cluster deployment is the Hadoop Distributed File System (HDFS) for storing the data and MapReduce for determining data of value. Other Hadoop software components that are often part of a deployment include:

Flume
: Used for collecting and aggregating large amounts of log/event data on HDFS and deployed as a service

Fuse-DFS
: Enables integration with other systems for data import and export by allowing mounting of HDFS volumes using the Linux FUSE filesystem

HBase
: A columnar database with support of data summaries and ad hoc queries

Hive
: A SQL-like language, metadata repository, and data warehousing framework with a rudimentary rules-based optimizer for Hadoop

Mahout
: A machine learning and data mining programming library

Oozie
: A workflow engine and job scheduler for Hadoop

Pig
: A high level dataflow programming language and compiler for producing and executing MapReduce programs

Sqoop
: A tool used in transferring data between Hadoop and relational databases that uses MapReduce for import and/or export and supports direct data import into Hive tables

Zookeeper
: The coordination service for distributed applications running on Hadoop

Though MapReduce-like functionality is supported in Oracle Database 12*c* through pattern matching (as we will note later), most organizations will likely continue to deploy separate optimized Hadoop clusters when analyzing such data. The data warehouse greatly complements the Hadoop cluster platform as the relational database serves as a destination for the Big Data of value and provides a standard SQL interface for querying all data. In addition, the data warehouse is generally deployed for higher availability and better recovery than a Hadoop cluster, and provides higher levels of security than possible with Hadoop today. We will describe later in this chapter how the data warehousing topology often evolves when Big Data is included.

Data Warehouse Design

The database serves as the foundation of the business intelligence infrastructure: it is the place where the data is stored. But there is more to business intelligence than data

—the infrastructure becomes useful only when business users use the data to gain insight. This may seem like a trivial point, but we've seen numerous companies build elegant infrastructure without consulting business users to determine business needs or key performance indicators (KPIs) to be measured. Often, such deployed projects end up supporting very few users, generate little activity, and little business intelligence is gained.

Assuming that your infrastructure is well planned and there is a demand for the data, your next challenge will be to figure out how to handle the demand. You will be faced with the need to design your data warehouse and other infrastructure components to deliver appropriate performance to your users—performance that may initially seem far beyond your capabilities, since the information needed can involve comparisons of massive amounts of detailed data.

When you start your design, also remember that the data warehouse and business intelligence infrastructure will never be considered finished. When the business needs change, so too must components in the infrastructure. Thus, the ability to track changes through metadata stored in a repository often becomes critical as part of the design work. Various tools from Oracle and other vendors can provide this capability.

As noted previously, data warehouses historically have had a different set of usage characteristics from those of an OLTP database. One aspect that makes it easier to meet data warehousing performance requirements is the higher percentage of read operations. Oracle's locking model, described in detail in Chapter 8, is ideally suited for data warehouse operations. Oracle doesn't place any locks onto data that's being read, thus reducing contention and resource requirements for situations where there are a lot of database reads. Since locks don't escalate, Oracle is also frequently deployed where near real-time data feeds into the warehouse occur in a scenario not unlike OLTP workloads.

Warehousing usage characteristics lead to deploying different types of schema. In OLTP databases, transaction data is usually stored in multiple tables and data items are stored only once in what is called 3NF or third normal form (described in Chapter 4). If a query requests data from more than one transaction table, the tables are joined together. Typically, the database query optimizer decides which table to use as the starting point for the join, based on the assumption that the data in the tables is essentially equally important.

When business users need an understandable schema to formulate their own ad hoc queries or analytical processing is required, key transaction data can be more appropriately stored in a central fact table, surrounded by dimension or lookup tables, as shown in Figure 10-2. The fact table can contain summarized data for data items duplicated elsewhere in the warehouse, and dimension tables can contain multiple hierarchies. As noted previously, when organizations consolidate their data marts into enterprise data warehouses, many now deploy a variation called a hybrid schema, a mixture of third normal form and star schema.

Ralph Kimball, author of the widely read book *The Data Warehouse Toolkit* (Wiley), is largely credited with discovering that users of data warehouses typically pose their queries so that a star schema, illustrated in Figure 10-2, is an appropriate model to use. A typical query might be something such as the following:

> Show me how many sales of computers (a product type) were sold by a store chain (a channel) in Wisconsin (a geography) over the past 6 months (a time).

The schema in Figure 10-2 shows a relatively large sales transactions table (called a fact table) surrounded by smaller tables (called dimensions or lookup tables). The query just described is often called *multidimensional*, since several dimensions are included (and time is almost always one of them). Because these queries are typical in a data warehouse, the recognition of the star schema by Oracle's cost-based optimizer can deliver enormous performance benefits.

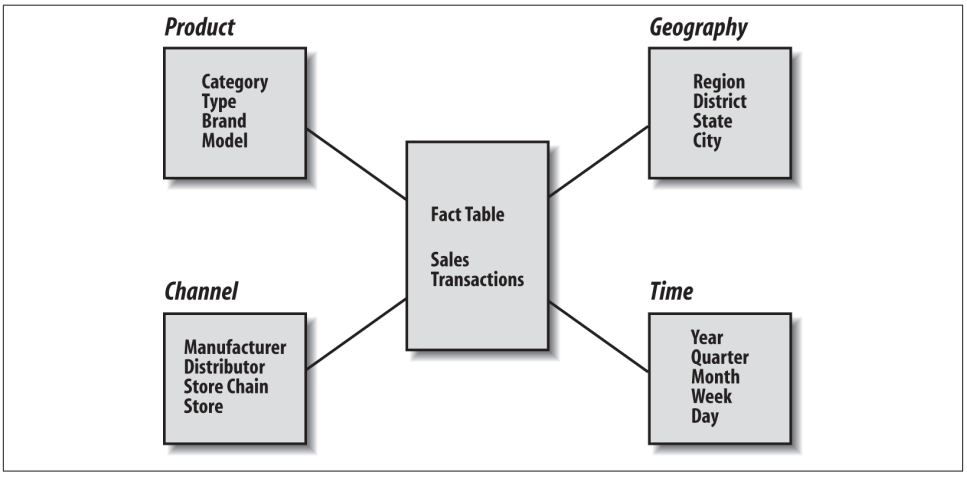


Figure 10-2. Typical star schema

Query Optimization

Oracle first provided the ability to recognize a star schema in the query optimizer in Oracle7 and has focused on making its cost-based query optimizer smarter in response to business intelligence queries in subsequent database releases. Since Oracle Database 10*g*, optimizer predictions are compared to actual runtime performance to improve optimizer prediction accuracy and any errors are subsequently corrected automatically. The optimizer also can provide query rewrite transparently to summary levels commonly deployed with star schema through materialized views, described later in this chapter. Oracle Database 11*g* added query rewrite for the OLAP Option as well as improved solving of queries containing inline views.

In Oracle Database 12*c*, during the compilation of a SQL statement, the optimizer automatically decides whether available statistics can generate a good execution plan. If statistics are missing or out of date, dynamic sampling of tables automatically occurs to generate new statistics. Query performance is further improved by adaptive execution plans that make corrections during execution.

How does the optimizer handle a query against a star schema? First, it finds a sales transactions fact table (shown in Figure 10-2) with a lot more entries than the surrounding dimension tables. This is the clue that a star schema exists. Early databases would have tried to join each of the dimension tables to the fact table, one at a time. Because the fact table is usually very large, using the fact table in multiple joins takes a lot of computer resources.

Cartesian product joins were added to Oracle7 to first join the dimension tables, with a subsequent single join back to the fact table in the final step. This technique works relatively well when there are not many dimension tables (typically six or fewer, as a rule of thumb, to keep the Cartesian product small) and when data is relatively well populated.

In some situations, there are a fairly large number of dimension tables or the data in the fact table is sparse. For joining such tables, a parallel bitmap star join may be selected by the optimizer.

In earlier releases of the Oracle Database, DBAs had to set initialization parameters (e.g., STAR_TRANSFORMATION) and gather statistics, enabling the optimizer to recognize the best methods for solving such queries. Today, needed parameters are preset upon installation and statistics are automatically gathered by the Oracle Database.

Bitmap Indexes and Parallelism

Bitmap indexes, described in Chapter 4, were first introduced in Oracle7 to speed up the type of data retrieval and joins in data warehousing queries. Bitmap indexes in Oracle are typically considered for columns in which the data has low cardinality. *Cardinality* is the number of different values in an index divided by the number of rows. There are various opinions about what low cardinality actually is. Some consider cardinality as high as 10% to be low, but remember that if a table has a million rows, that "low" cardinality would mean 100,000 different values in a column!

In a bitmap index, a value of 1 in the index indicates that a value is present in a particular row, and 0 indicates that the value is not present. A bitmap is built for each of the values in the indexed columns. Because computers are built on a concept of 1s and 0s, this technique can greatly speed up data retrieval. In addition, join operations such as AND become a simple addition operation across multiple bitmaps. A side benefit is that bitmap indexes can provide considerable storage savings.

Figure 10-3 illustrates the use of a bitmap index in a compound WHERE clause. Bitmap indexes can be used together for even faster performance. The bitmap indexes are essentially stacked together, as a set of punch cards might be. Oracle simply looks for those parts of the stack with all the bits turned on (indicating the presence of the value), in the same way that you could try to stick a knitting needle through the portions of the card stack that were punched out on all of the cards.

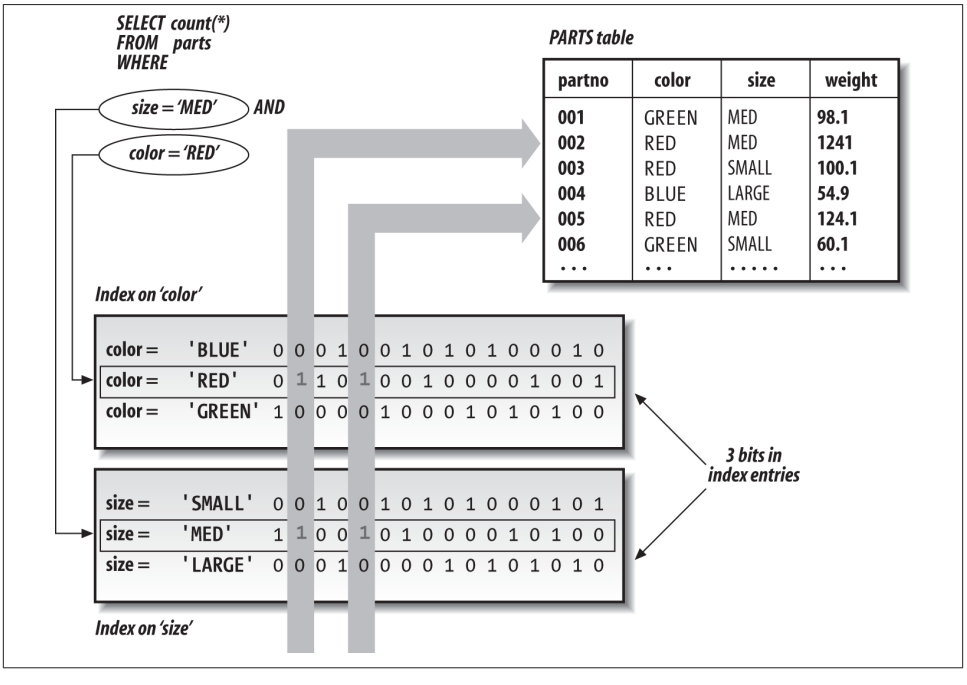


Figure 10-3. Bitmap index operation in a compound WHERE clause

In Oracle, star-query performance is improved when bitmap indexes are created on the foreign-keys columns of the fact table that link to the surrounding dimension tables. A parallel bitmap star join occurs in which the bitmaps retrieve only the necessary rows from the fact table and the rows are joined to the dimension tables. During the join, sparseness (i.e., a large quantity of empty values) is recognized inherently in the bitmaps, and the number of dimension tables isn't a problem. This algorithm can also efficiently handle a *snowflake schema*, which is an extension of a standard star schema in which there are multiple tables for each dimension.

To further speed queries, Oracle9*i* added a bitmap join index from fact tables to dimension tables. A bitmap join index is simply the bitmap index of a join of two or more tables. The speedup in performance comes from avoiding actual table joins or reducing the amount of data joined by taking into account restrictions in advance of the joining

of data. Performance speedup for star queries with multiple dimension tables can be greatly improved since bitwise operations in star transformations can now be eliminated.

Performing queries in parallel also obviously improves performance. Joins and sorts are frequently used to solve business intelligence queries. Parallelism is described in Chapter 7. That chapter lists functions that Oracle can perform in parallel (see the section "What Can Be Parallelized?" on page 187).

Real Application Clusters, which replaced Oracle Parallel Server as of Oracle9*i*, further expands parallelism by enabling queries to transparently scale across nodes in clusters or in grids of computer systems.

Since Oracle Database 10*g*, statistics gathering automatically populates the Automatic Workload Repository, an important source of information for Oracle's cost-based optimizer. For example, the SQL Access Advisor leverages this information when making tuning recommendations. Oracle Database 12*c* provides support for adaptive plans, query plans that are adjusted at run-time based on current data. Adaptive statistics at run-time and compile-time in Oracle Database 12*c* enable optimization "learning" for future queries.

Optimization Provided by the Exadata Storage Server Software

The introduction of the *Oracle Exadata Database Machine* as an engineered system in 2008 enabled Oracle to define balanced hardware server and storage configurations that are linked via a high-speed interconnect (InfiniBand). Today, Exadata contains powerful Sun server components including those that provide the Exadata Storage Server cells. The Exadata Storage Server software enables the Oracle Database to perform unique query and analytics optimization in Exadata storage transparently to the applications.

Smart Scans offload query predicate evaluation to storage, performing row, column, and join filtering in the Exadata Storage Server cells. During star join filtering provided by Smart Scans, dimension table predicates are transformed into filters that are applied to the fact table scan. Storage indexes in the Exadata Storage Server software further assist by transparently keeping track of minimum and maximum values of columns stored in tables on a cell, eliminating scan I/O if a WHERE clause in the query is outside the bounds of those values. The optimizations are important in that they minimize both the occurrence of much slower full table scans and the need for extensive indexing to eliminate such full table scans.

Smart Scans transparently handle a variety of complex cases including uncommitted data and locked rows, chained rows, compressed tables, national language processing, date arithmetic, regular expression searches, and partitioned tables. Smart Scans also

help in more efficient backups (since only changed blocks are returned from storage), more efficient creation of Oracle tablespaces, and can process Encrypted Tablespaces (TSE) and Encrypted Columns (TDE) in storage. Data mining scoring and statistics are also processed in Exadata Database Machine storage, beneficial since the number of processors in the Exadata Storage Server cells exceed the number of processors in the Database Server nodes.

The Exadata Storage Server software supports Hybrid Columnar Compression (HCC). HCC is most often associated with storage volume savings since typical compression of data warehouse data is about 8 to 10 times and compression of archive data is up to about 15 times. But there are some benefits during query processing as well. Queries run against compressed data do not require decompression in storage during Smart Scans, reducing the number of I/Os to scan a table. Only columns and rows that satisfy a given predicate are decompressed in memory on the Exadata Storage Server cell.

The Exadata Storage Server cells contain flash, which is used, by default, as a Smart Flash Cache by the Oracle Database. In a data warehouse, the Smart Flash Cache can speed performance as it transparently stores data that will be reused in other queries in the flash. Administrators can also direct at the table, index, or segment level what data should be retained in flash. The Smart Flash Cache can also reduce log write I/O latency, more often associated with transaction processing systems. However, for data warehouses, this reduction can also be beneficial during ETL processing.

Dimensional Data and Hierarchies in the Database

Next we'll take a look at how the Oracle Database supports the concepts of dimensions and hierarchies in a bit more detail with a focus on summary tables, materialized views, and the Oracle OLAP Option.

Summary Tables

Data within dimensions is usually hierarchical in nature (e.g., in the time dimension, day rolls up to week, which rolls up to month, which rolls up to quarter, which rolls up to year). If the query is simply looking for data summarized at a monthly level, why should it have to sort through more detailed daily and weekly data? Instead, it can simply view data at or above that level of the hierarchy. Formerly, data warehousing performance consultants designed these types of summary tables—including multiple levels of pre-calculated summarization. For example, all the time periods listed in Figure 10-2 can be calculated on the fly using different groupings of days. However, to speed queries based on a different time series, a data warehouse can have values pre-calculated for weeks and months and stored in summary tables to which queries can be redirected.

Materialized Views

Oracle8*i* introduced the concept of *materialized views* for the creation of summary tables for facts and dimensions that can represent rollup levels in the hierarchies. A materialized view provides pre-computed summary data; most importantly, a materialized view is automatically substituted for a larger detailed table when appropriate. The cost-based query optimizer can perform query rewrites to these summary tables and rollup levels in the hierarchy transparently, often resulting in dramatic increases in performance. For instance, if a query can be answered by summary data based on sales by month, the query optimizer will automatically substitute the materialized view for the more granular table when processing the query. A query at the quarter level might use monthly aggregates in the materialized view, selecting the months needed for the quarter(s). Oracle Database 10*g* added query rewrite capabilities such that the optimizer can make use of multiple appropriate materialized views.

Materialized views can be managed through Oracle Enterprise Manager (see also Chapter 5). The SQL Advisor accessible in Enterprise Manager includes a SQL Access Advisor that can recommend when to create materialized views.

OLAP Option

As business users become more sophisticated, their questions evolve from "what happened" to "what trends are present and what might happen in the future?" *OLAP tools* provide the ability to handle time-series and mathematical analysis for understanding past trends and forecasting the future.

OLAP initially grew around the early inability of relational databases to effectively handle multidimensional queries (described previously in the section "Data Warehouse Design" on page 250). This led to OLAP tools packaged with their own data "cubes" where data is downloaded from relational sources into the cubes.

These separate database engines are called Multidimensional Online Analytical Processing engines, or *MOLAP engines*. Oracle Essbase is one such example. This standalone OLAP solution (e.g., not part of the Oracle Database OLAP Option) is especially popular for Hyperion's financial applications and in cases where business analysts want to generate their own cubes. Essbase cubes can also be accessed using Oracle Business Intelligence Enterprise Edition (OBI EE) and spreadsheet tools. Such MOLAP engines handle queries extremely quickly and work best when the cube is not updated frequently (because the cube-generation process takes time). Data can be gathered from a variety of database sources.

Within relational databases, OLAP functionality also appeared since star schema containing summary levels are supported to various degrees. When used in this fashion, the interaction is called *ROLAP*, which stands for Relational Online Analytical

Processing. Tools that can work against either relational databases or MOLAP engines are sometimes referred to as *hybrid tools*.

The Oracle OLAP Option can be thought of as a MOLAP cube within the relational database, accessible via SQL. Though available since Oracle 9*i*, Oracle Database 11*g* significantly improved the flexibility of accessing the OLAP Option. Business users formerly needed to specifically point their queries to OLAP Option cubes. Since Oracle Database 11*g*, the OLAP cubes can be used transparently as an alternative to materialized views because Oracle's SQL query rewrite recognizes the cubes. The materialized view refresh can refresh OLAP cubes as of Oracle Database 11*g*.

OLAP Option cubes are deployed in what are called *analytic workspaces*. They can be created using a simplified logical dimensional modeling tool called the Analytic Workspace Manager (AWM). The tool provides an interface for creation of the cubes and for building maps from relational tables into the cubes.

Custom OLAP applications can be built using Oracle's JDeveloper and *business intelligence beans*, although this is much less common than using off-the-shelf tools. The Java beans provide prebuilt components for manipulating tables, crosstabs, and graphs, and for building queries and calculations similar to the functionality previously found in Express. JDeveloper generates Java code utilizing these building blocks that maps to the Java OLAP API provided by Oracle's OLAP Option.

Analytics and Statistics in the Database

Analysis of large data sets is faster when it takes place where the data is stored, since this approach avoids the overhead of moving the data sets around. This section describes the database functions and other features available for analytics, statistics, and data mining in the Oracle Database.

It is worth noting here that the growing use of Oracle for statistical computations led to support for floating-point number types providing the precision outlined in the IEEE 754–1985 standard (with minor differences). These are provided in the datatypes BINARY_FLOAT and BINARY_DOUBLE in Oracle Database 10*g* and more recent database releases.

Basic Analytic and Statistical Functions

Oracle releases dating back to Oracle8*i* have continued to add new analytic and statistical functions as SQL extensions to the core Oracle Enterprise Edition and Standard Edition databases. These analytic functions now include:

Ranking functions
: Used to compute a record's rank with respect to other records. Functions include RANK, DENSE_RANK, CUME_DIST, PERCENT_RANK, NTILE, and ROW_NUMBER. Hypothetical ranking is also supported.

Windowing and Reporting aggregate functions
: Used to compute cumulative and moving averages. Functions include SUM, AVG, MIN, MAX, COUNT, VARIANCE, STDDEV, FIRST_VALUE, LAST_VALUE, and RATIO_TO_REPORT.

LAG/LEAD functions
: Often used to compare values from similar time periods, such as the first quarter of 2013 and the first quarter of 2012.

Linear regression functions
: Include REGR_COUNT, REGR_AVGX and REGR_AVGY, REGR_SLOPE, REGR_INTERCEPT, REGR_R2, and other functions used in regression line fitting for a set of numbers in pairs (e.g., having X and Y values).

Also supported in Oracle are pivoting operations, histograms (using WIDTH_BUCKET), CASE expressions, filling gaps in data, and time-series calculations.

The database includes a statistics package, DBMS_STATS_FUNCS. Functions in the statistics package support linear algebra, frequent itemsets, descriptive statistics, hypothesis testing (T-test, F-test, Binomial test, Wilcoxon Signed Ranks test, One-Way ANOVA, Chi-square, Mann Whitney, Kolmogorov-Smirnov), crosstab statistics (% statistics, Chi-squared, phi coefficient, Cramer's V, contingency coefficient, and Cohen's kappa), and nonparametric correlation (Pearson's correlation coefficients, and Spearman's and Kendall's).

Other SQL Extensions

The SQL MODEL clause first appeared in Oracle Database 10*g* as an extension to the SELECT statement. This clause enables relational data to be treated as multidimensional arrays (much like spreadsheets) and is also used to define formulas for the arrays, avoiding multiple joins and UNION clauses.

MODEL supports analytical queries that include prior-year comparisons and mathematical business rules and it is particularly useful in budgeting, forecasting, and other statistical applications. Example MODEL usages include calculating sales differences in two geographies, calculating percentage change, and calculating net present value. The SQL MODEL clause can also use simultaneous equations and regression in calculations.

Oracle Database 12*c* introduces SQL pattern matching used in finding patterns in data across multiple rows. It is particularly useful when looking for repeating sequences in data, such as for when you might be trying to identify unusual behavior or are

determining when to make an investment. This function is sometimes described as providing MapReduce-like functionality in the Oracle Database as it greatly simplifies pattern-matching coding and the maintenance of such code.

The MATCH_RECOGNIZE clause typically is used to perform a PARTITION BY to identify a data item of interest (such as a company name) and ORDER BY to order each row partition. Then it will search each row for matches in a defined PATTERN by incrementally looking for the match row by row. It calculates the row pattern measure columns after a match is found (as defined by MEASURES). It will report ONE ROW PER MATCH or ALL ROWS PER MATCH depending on which is specified, and uses AFTER MATCHES SKIP to determine where to continue to look for row pattern matches after a match is found.

Advanced Analytics Option

The Advanced Analytics Option consists of Oracle R Enterprise and Oracle's former Data Mining Option. This option, first available in 2012, enables advanced statistics and data mining algorithms to be applied to data residing in an Oracle Enterprise Edition Database.

R is the increasingly popular open source programming language for statistical analysis and graphical display. First developed in 1994, R became popular in universities in the past decade and there are now over 2 million users.

Oracle R Enterprise is an embedded component of the Oracle Database available for Linux and several other popular operating systems. Workstation memory constraints and scalability are removed since R scripts and the development environment (RStudio) take advantage of the power of the database platform during data preparation, model development, and model deployment. Oracle R Enterprise includes an R-SQL package transparency framework providing transparent database table access and in-database execution, a database library statistics engine, and SQL extensions enabling in-database execution of R code. Any CRAN open source packages can be run in the database either via Oracle R Enterprise to SQL function pushdown or in native R mode. The Oracle Database statistics engine includes R support for:

- Density, probability, and quantile functions
- Special functions (such as Gamma functions)
- Tests (such as Chi-square, simple & weighted kappas, and correlation)
- Base SAS equivalents (such as frequency, summary, sort, rank, and others)

Data mining, an often overused and misunderstood term in data warehousing, is the use of mathematical algorithms to model relationships in the data that wouldn't be apparent by using other tools. Most companies shouldn't approach data mining unless analysts have met the following criteria:

- An understanding of the quality and meaning of the data in the warehouse.
- Business insight gained using other tools and the warehouse.
- An understanding of a business issue being driven by too many variables to model outcomes in any other way.

In other words, data mining tools are not a replacement for the analytical skills of data warehouse users.

The data mining tools themselves can rely on a number of techniques to produce the relationships, such as:

- Extended statistical algorithms, such as those provided by R and other statistical tools, to highlight statistical variations in the data.
- Clustering techniques that show how business outcomes can fall into certain groups, such as insurance claims versus time for various age brackets. In this example, once a low-risk group is found or classified, further research into influencing factors or "associations" might take place.
- Logic models (if A occurs, then B or C is a possible outcome) validated against small sample sets and then applied to larger data models for prediction, commonly known as *decision trees*.
- Neural networks "trained" against small sets, with known results to be applied later against a much larger set.
- Anomaly detection used to detect outliers and rare events.
- Visualization techniques used to graphically plot variables and understand which variables are key to a particular outcome.

Data mining is often used to solve difficult business problems such as fraud detection and churn in micro-opportunity marketing, as well as in other areas where many variables can influence an outcome. Companies servicing credit cards use data mining to track unusual usage—for example, the unexpected charging to a credit card of expensive jewelry in a city not normally traveled to by the cardholder. Discovering clusters of unusual buying patterns within certain small groups might also drive micro-opportunity market campaigns aimed at small audiences with a high probability of purchasing products or services.

Oracle first began to embed algorithms packaged as the Data Mining Option into the Oracle9*i* database. Algorithms now in the Advanced Analytics Option include Naïve Bayes, Associations, Adaptive Bayes Networks, Clustering, Expectation Maximization (EM), Support Vector Machines (SVM), Nonnegative Matrix Factorization (NMF), Decision Trees, Generalized Linear Models (supporting Binary Logistic Regression and Multivariate Linear Regression), Principal Component Analysis (PCA), and Singular

Value Decomposition (SVD). The algorithms are accessible via Java and PL/SQL APIs. Other data mining capabilities available include text mining (providing document clustering and classification) and BLAST similarity searches leveraging the SVM algorithms (common in genetic research).

Data mining applications can be custom-built using Oracle's Data Miner tool. Data Miner is used to develop, test, and score the models. The data is usually prepared for mining by binning, normalizing, and adjusting for missing values in the Oracle Database. Data Miner also provides the ability to define metadata, tune the generated Java code, view generated XML files, and test application components.

Other Datatypes and Big Data

A number of other types of data are often stored in an Oracle Database. Among the types of data are:

Multimedia and Images
: The Multimedia feature set (once known as *inter*Media) opens up the possibilities of including documents, audio, video, and some locator functions in the warehouse. Of these, text retrieval (Oracle Text) is most commonly used in warehouses today. However, the number of organizations storing other types of data, such as images, is growing. Oracle first added DICOM imaging support, popular in medicine, to Oracle Database 11*g*. Often, storage of these types of data is driven by a need to provide remote users with access.

Spatial Data
: The spatial locator capability in the Oracle Database enables retrieval of data based on a geo-spatial location. Where distances and other geographic computations are needed, the Spatial Option enables such applications to be built. An example of this option's use for data warehousing is a marketing analysis application that determines the viability of retail outlets at various locations.

XML
: Oracle first added native XML datatype support to Oracle9*i*, along with XML and SQL interchangeability for searching. Oracle provided key technology in the development of the XQuery standard, and began shipping a production version of XQuery with Oracle Database 10*g* Release 2. XML database performance was greatly improved in Oracle Database 11*g* through the introduction of binary XML.

Though most of this book describes SQL as the primary way to access data, various programming paradigms remain popular. As noted earlier in this chapter, MapReduce is gaining in popularity for processing semi-structured data stored in Hadoop using callouts from programming languages such as Java and Python. In some situations where the organization would prefer to store all of the data in an Oracle Database and

most of the data to be analyzed is structured, Oracle Database 12*c* pattern matching capabilities might suffice for processing the semi-structured data.

Where huge data volumes of semi-structured or unstructured data are being gathered, separate Hadoop clusters are used to filter data and MapReduce results are often loaded into an Oracle data warehouse. Oracle offers an engineered system for deploying Hadoop clusters called the *Oracle Big Data Appliance* (BDA). A Full Rack consisted of 18 nodes, 648 terabytes of disk, 1,152 GB of memory, 288 processing cores, and an InfiniBand interconnect when this edition of *Oracle Essentials* was published. Starter Racks populated with 6 nodes were also available. Multiple Full Racks can be connected via InfiniBand using internal switches in the Racks. The foundation software for the BDA includes the Cloudera Distribution of Hadoop (CDH), Cloudera Manager, Oracle NoSQL Database Community Edition, Java VM, and Linux. In a standard configuration, data is distributed across the entire platform using HDFS and triple replicated. As with other engineered systems, Oracle provides a single point of support for the entire configuration.

Loading Data into the Data Warehouse

Experienced data warehouse architects realize that the process of understanding the data sources, designing transformations, testing the loading process, and debugging is often the most time-consuming part of deployment. Transformations are used to remove bogus data (including erroneous entries and duplicate entries), convert data items to an agreed-upon format, and filter data not considered necessary for the warehouse. These operations are often used to improve the quality of data loaded into the warehouse.

The frequency of data extraction from sources and loading into the data warehouse is largely determined by the required timeliness of the data in order to make business decisions. Most data extraction and loading takes place on a "batch" basis and data transformations cause a time delay. Early warehouses were often completely refreshed during the loading process, but as data volumes grew, this became impractical. Today, updates to tables are most common. When a need for near real-time data exists, warehouses can be loaded nearly continuously using a *trickle feed* if the source data is relatively clean, eliminating the need for complex transformations. If real-time feeds are not possible but real-time recommendations are needed, engines such as Oracle's Real-time Decisions are deployed.

Is Cleanliness Best?

Once the data in the warehouse is "clean," is this version of the true nature of the data propagated back to the originating OLTP systems? This is an important issue for data warehouse implementation. In some cases, a "closed loop" process is implemented

whereby updates are provided back to the originating systems. In addition to minimizing some of the cleansing that takes place during future extractions, operational reports become more accurate.

Another viable option is to avoid cleansing by improving the quality of the data at the time of its input into the operational system. As noted previously in this chapter, this is critical if OLTP systems are to be directly accessed for business intelligence. Improving data quality at the source also enables high-speed loading techniques to be used in near real-time data warehouses (since transformations can be eliminated).

Improving data quality at the source can sometimes be accomplished by not allowing a "default" condition as allowable input into a data field. Presenting the data-entry person with an array of valid options, one of which *must* be selected, is often a way to ensure the most consistent and valid responses. Many companies also provide education to the data-entry people, showing them how the data they're keying in will be used and what the significance of it is.

Key Oracle products and database features that often help facilitate this process include:

Oracle Data Integrator (ODI)
: Acquired by Oracle in 2007, this extraction, transformation, and loading (ETL) tool that handles heterogeneous sources and targets is sometimes referenced as an ELT tool since transformations are pushed into the target data warehouse. This product has replaced Oracle Warehouse Builder as Oracle's primary offering for ETL. ODI features Knowledge Modules that define integration capabilities, including extraction with change data capture, loading and unloading utilities, SQL-based loading and unloading, and transformation logic SQL. Data Quality options include data quality profiling, batch processing, and address verification. The Knowledge Modules are modifiable. The product architecture includes a development environment that makes use of the Knowledge Modules as templates in declarative design processes and an orchestration agent. ODI can be used for data-based, event-based, and service-based data integration.

Oracle GoldenGate
: Acquired by Oracle in 2009, GoldenGate has replaced Oracle Streams as the primary software recommended for log-based replication. Often used for zero downtime software upgrades, during software migrations, and for low latency transaction replication and recovery, GoldenGate supports a wide variety of data sources and targets. It is often used to load Oracle-based data warehouses where the need for data transformations is minimal and near real-time updates of the data in the data warehouse are desired.

Transparent Gateways and Heterogeneous Services
: Provide a bridge to retrieve data from non-Oracle sources using Oracle SQL to load an Oracle Database. Heterogeneous Services provide ODBC connectivity to non-

Oracle relational sources. Gateways can optionally provide a higher level of performance when extracting data from non-Oracle sources.

Transportable Tablespaces
: Another feature for data movement, Transportable Tablespaces enable rapid data movement between Oracle instances without export/import. Metadata (the data dictionary) is exported from the source and imported to the target. The transferred tablespace can then be mounted on the target. Oracle Database 10*g* introduced cross-platform Transportable Tablespaces, which can move a tablespace from one type of system (e.g., Solaris) to another (e.g., Linux).

Transportable Partitions
: Oracle Database 11*g* introduced Transportable Partitions for rapid data movement between Oracle instances.

Data Pump Fast Import/Export
: Added in Oracle Database 10*g* and enabled via external table support, Data Pump is a newer import/export format. Parallel direct path loading and unloading are supported.

Oracle Big Data Connectors
: First available in 2011, Oracle's Big Data Connectors include an Oracle Loader for Hadoop that pushes the preprocessing of data to be loaded into an Oracle data warehouse to the source Hadoop Big Data cluster. The result is lessened CPU and I/O impact on the target Oracle Database platform. An Oracle Data Integrator Application Adapter for Hadoop provides ODI Knowledge Modules optimized for Hive and the Oracle Loader for Hadoop. An Oracle Direct Connector for HDFS and an Oracle R Connector for Hadoop are also provided.

The Oracle Database helps the ETL and loading process in a variety of ways. For high-speed loading of flat files, Oracle SQL*Loader's *direct path load* option provides rapid loading by bypassing the buffer cache and rollback mechanism and writing directly to the datafile. You can run SQL*Loader sessions in parallel to further speed the table-loading process (as many warehouses need to be loaded in a limited "window" of time). Many popular ETL tools, including ODI, generate SQL*Loader scripts.

The core Oracle Database engine features embedded ETL functions that ETL tools support to varying degrees. Examples of these features include support for external tables, table functions, merge (i.e., insert or update depending on whether a data item exists), multitable inserts, change data capture, and resumable statements.

Managing the Data Warehouse

Oracle Enterprise Manager (EM) provides a common GUI for managing your multiple database instances regardless of the underlying operating system and server platform

you choose. EM is browser-based with a multiuser repository for tracking and managing the Oracle instances. (EM is discussed in much more detail in Chapter 5.)

In warehousing, in addition to basic management, ongoing tuning for performance is crucial. Enterprise Manager supports many of the automated diagnostics and tuning features added in recent database releases.

Within the largest warehouses and data marts, you may want to manage or maintain availability to some of the data even as other parts of the database are moved offline. Oracle's Partitioning Option enables data partitions based on business value ranges (such as date) or discrete values for administrative flexibility, while enhancing query performance through the cost-based optimizer's ability to eliminate access to nonrelevant partitions. For example, "rolling window" administrative operations can be used to add new data and remove old data using time ranges. A new partition can be added, loaded, and indexed in parallel, and optionally removed, all without impacting access to existing data.

Range partitioning first became available in the Oracle8 Partitioning Option. *Hash partitioning* was added to the Oracle8*i* Partitioning Option, enabling the spread of data evenly based on a hash algorithm for performance. Hashing may be used within range partitions (*composite partitioning*) to increase the performance of queries while still maintaining the manageability offered by range partitioning. Oracle9*i* introduced *list partitioning*—partitions based on discrete values such as geographies. A composite partitioning type, *range-list partitioning*, which allows you to partition by dates within geographies, was added in Oracle9*i* Release 2. More composite types were added in Oracle Database 11*g* including *list-hash*, *list-list*, *list-range*, *range-range partitioning*, *(parent-child) reference partitioning*, and *virtual column partitioning*. *Interval partitioning*, also added in Oracle Database 11*g*, provides automatic creation of range partitions when needed. Oracle Database 12*c* added *interval-reference partitioning* to the mix of available partitioning types.

Data Warehouses and Backups

Early data warehousing practitioners often overlooked the need to perform backups. Their belief was that since data for the warehouse was extracted from operational systems, the warehouses could easily be repopulated from those same systems if needed. However, as warehouses grew and the transformations needed to create and refresh them evolved, it became evident that backups of data warehouses were necessary because the transformation process had grown extremely complicated and time-consuming. Today, planning for warehouse availability includes not only an understanding of how long loading will take, but also backup and recovery operations. Due to the tactical nature of such warehouses, planning often also includes designs for high availability, disaster recovery, and information lifecycle management.

The approach of overlooking these needs could be repeating itself in Big Data platform deployment strategies today. As Hadoop advances, potentially rapidly, tools for managing and recovering data as the volumes continue to grow on the Hadoop clusters will assume a greater importance in day-to-day operations.

Business Intelligence Tools

Marketing, financial, and other business analysts are rarely interested in the storage and schema that hold their information. Their interest level rises when the discussion turns to the tools they'll be using to help them make intelligent decisions based on that information. Business intelligence tools are often evaluated and purchased within individual business areas, sometimes without close IT coordination. When business analysts want to do query, reporting, and analysis against data residing in an Oracle data warehouse, you might choose the Oracle Business Intelligence Foundation Suite or popular independent vendors' products, such as SAP Business Objects, IBM Cognos, MicroStrategy, and others.

A new class of tools has emerged in a category named data discovery tools. These tools are used for exploring data in schema-less data stores, extracting data from relational sources such as the Oracle Database and semi-structured data sources, such as Hadoop. One such example is Oracle Endeca.

In this section of the chapter, we will describe in more detail Oracle's business intelligence tools and applications, and Oracle's data discovery tools.

Oracle Business Intelligence Foundation Suite

The Oracle business intelligence tools that are most often selected today reside in Oracle's Business Intelligence Foundation Suite. The Suite includes middle-tier server components and business analyst delivery components. Key server components include:

Oracle BI Server
: This mid-tier query and analysis server provides linkage to data from Oracle Databases, other relational databases, semi-structured data sources, and OLAP technology. It can be used when federating data from multiple sources and supports query caching. A Common Enterprise Information Model provides a semantic model that consists of a physical layer (e.g., tables, columns, joins, and security parameters when describing an Oracle Database), a business model and mapping layer, and a presentation layer.

Oracle Essbase
: This is a multidimensional online analytical processing (MOLAP) engine that can be accessed using Oracle's BI tools and Hyperion applications. Essbase features an extensive calculation language that supports conditional and logical operations,

Boolean functions, relationship functions, calculation operations, mathematical functions, member set functions, range and financial functions, allocation functions, forecasting functions, statistical functions, and date and time functions.

Key business intelligence components that deliver information to business analysts include:

Enterprise Reporting and Publishing (BI Publisher)
: Template-based publishing solution that incorporates XML data extracts and produces reports in various output formats including PDF, RTF, HTML, Excel, XML, and eText. Report editors include popular desktop tools such as Adobe Acrobat and Microsoft Word.

Ad hoc Query and Reporting
: A thin client interactive tool used for generating ad hoc queries and analysis, it can be used directly against relational databases and MOLAP data stores. Generated reports can be posted to the dashboard or serve as input to BI Publisher.

Scorecard and Strategy Management
: A highly visual environment used for defining KPIs and then creating and managing strategy maps, cause and effect diagrams, and custom views.

Interactive Dashboards
: The dashboards are an interactive web-based collection of content provided by the other BI components that can be highly customized. Guided navigation links can be set up to guide business analysts to more relevant dashboard content depending on the business situation. Briefing books consisting of report decks can also be defined using the dashboards.

Action Framework
: The Action Framework enables invocation of workflows, Web Services, Java methods, and other custom procedures in response to predefined business events and/or data exceptions. An alerting engine can also capture and distribute notifications.

Oracle BI Mobile
: Access to the BI dashboards from popular mobile devices (e.g., running iOS) is achievable with no reprogramming involved. As this book was published, the most popular form factor and device for accessing the dashboards was the Apple iPad.

Microsoft Office Integration
: The BI Office plug-in enables the embedding of data from Interactive Dashboards, Ad hoc Query and Reporting, BI Publisher, and the BI Server into Microsoft Word, Excel, and PowerPoint.

Figure 10-4 illustrates a typical dashboard in the Oracle Business Intelligence tools.

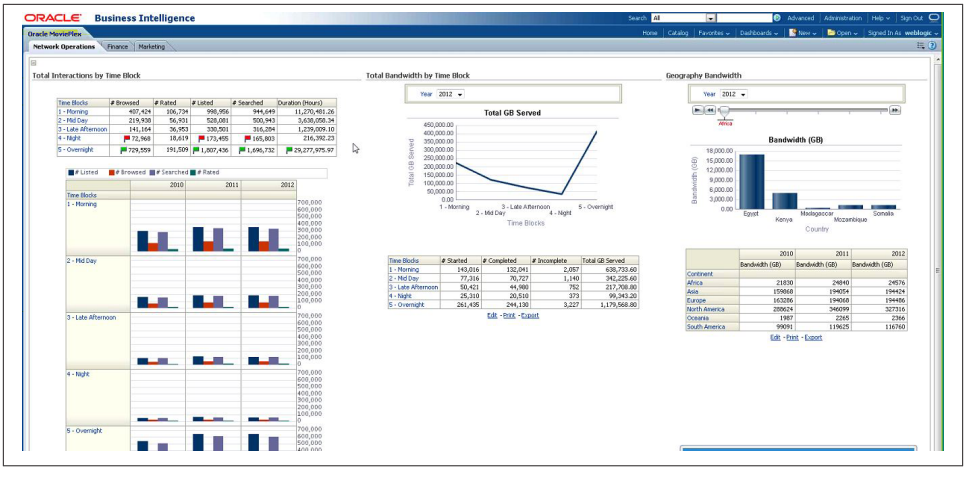


Figure 10-4. Oracle Business Intelligence Dashboard

Oracle has an enterprise portal offering, WebCenter, available as part of the Oracle Application Server, providing an integration point for custom-built business intelligence applications using Oracle Business Intelligence tools. For example, the BI Presentation Services offer an interface using the Simple Object Access Protocol (SOAP) to embed results into WebCenter. WebCenter can also enable social collaboration using the BI content.

Business Intelligence Applications

Business intelligence applications are prebuilt solutions providing extended reporting and "dashboard-like" interfaces to display business trends. These applications directly access data warehouse schema and sometimes also blend in transactional data sources. The data warehousing type of solutions can include prebuilt ETL from known transactional data sources. The business intelligence applications often focus on specific areas of the business, such as marketing or financial analysis, or industry data models in areas such as communications, finance, healthcare, and retail.

Oracle Business Intelligence Applications include star schema data models with conformed dimensions deployed to relational databases such as Oracle, more than 2,500 KPIs displayed in predefined dashboards, and feature prebuilt ETL mappings from Oracle Fusion Applications, Siebel CRM, Oracle E-Business Suite, PeopleSoft, JD Edwards, SAP, and other applications. The applications cover the areas of Sales, Service and Contact Center, Marketing, Financial, Supply Chain, Projects, and Workforce. Oracle is continuing to extend the KPIs provided, the ETL mappings, and the business areas covered.

The Oracle Hyperion Financial Performance Management applications address financial planning and budgeting and are deployed to Essbase servers. Such applications include predefined queries, reports, and charts that deliver the kind of information required for a particular type of business analysis while sparing the business user the complexity of creating these objects from scratch.

The promise of such prebuilt solutions is that they provide easier-to-deploy solutions with more out-of-the-box functionality. While some customization will probably always be needed, the time required to deploy an initial and useful solution can be substantially reduced.

Data Discovery and Oracle Endeca Information Discovery

Data discovery tools are gaining interest in business communities that want to explore data without waiting for IT to build a data model and are particularly useful in qualitative analysis. Oracle's *Endeca* Server provides a multifaceted key/value store into which you can load structured data (such as that stored in an Oracle Database) and semi-structured data (such as XML data, text, and Big Data). Loading is via the product's Integration Suite.

Business analysts using the Endeca Studio interface can perform deep searches across all of the data. Navigation through the data is based on data context, meaning the analyst browses through the data without any predefined paths. Data is displayed in charts, crosstabs, as key metrics, as geospatial information, and as tag clouds. Figure 10-5 illustrates a typical view in Endeca Studio.

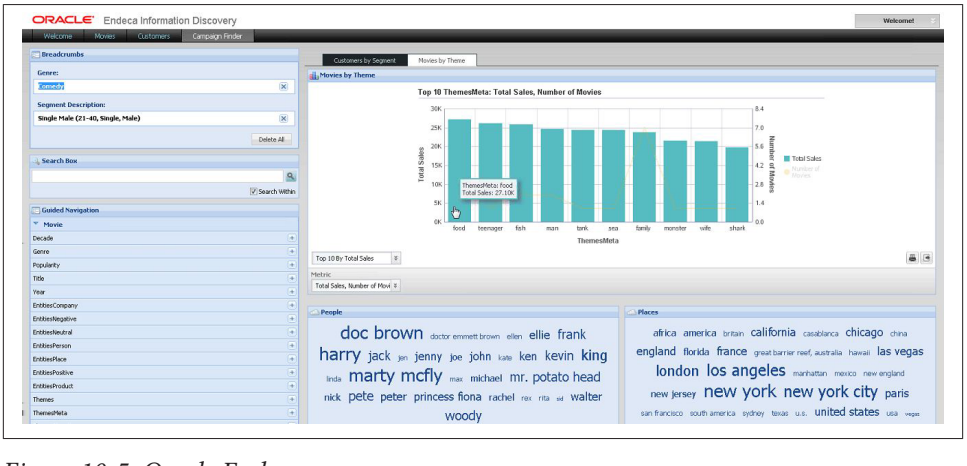


Figure 10-5. Oracle Endeca

Some organizations use the Endeca Server in lieu of building sandboxes in their data warehouse. Once business value has been identified using Endeca Studio, they

incorporate the same data feeds in the warehouse, build out that schema appropriately, and provide access to KPIs and data via their standard business intelligence tools.

Oracle Exalytics

Oracle Exalytics is an engineered system designed to run the Oracle Business Intelligence Suite, Endeca Information Discovery, and Hyperion Planning applications. It is a rack mountable unit that contains 40 processing cores and terabytes of memory. The large number of cores delivers a unique visualization experience for the Oracle Business Intelligence tools where results are returned as a mouse is moved across a dashboard without the need to hit a "go" button.

Performance speed-up for queries and discovery occurs because needed data and data structure is stored in memory. With the Oracle BI Server, results can automatically be cached in-memory or a summary advisor can be used to persist aggregates in-memory in the TimesTen for Exalytics database. For Essbase applications (including Hyperion), designated Essbase cube subject areas are stored in memory. For data discovery with Endeca, the Endeca Server is housed in memory.

The Metadata Challenge

On the one hand, *metadata*—or descriptive data about data—is incredibly important. Virtually all types of interactions with a database require the use of metadata, from datatypes of the data to business meaning and history of data fields.

On the other hand, metadata is useful only if the tools and clients who wish to use it can leverage it. One of the great challenges is to create a set of common metadata definitions that allows tools and databases from different vendors to interact.

There have been a number of attempts to reach an agreement on common metadata definitions. In 2000, a standard was ratified that defines a common interface for interchange of metadata implementations. Named the Common Warehouse Metadata Interchange (CWMI) by the Object Management Group (OMG), this standard is based on XML interchange. Oracle was one of the early proponents and developers of CWMI; however, there has been limited adoption and metadata exchange capabilities remain very vendor-specific today.

As noted earlier in this chapter, an emerging complementary solution—one in which ETL into a single data warehouse is not the entire solution—is the leveraging of master data management and data hub solutions. Today, most organizations are still a long way from consolidated metadata, and when they have tried to do this as an IT best practice project, they generally have not been successful. Such projects are usually adopted only when delivered within a business intelligence project that delivers business value.

Putting It All Together

Earlier in this chapter, we described the components in a complete end-to-end analytics architecture. Now, we'll take a look at what an end-to-end analytics infrastructure might look like with Oracle components for business intelligence, data warehousing, and Big Data. Then we'll discuss some best practices.

A Complete Analytics Infrastructure

How extensive an analytics infrastructure your organization needs depends on your business needs. At a minimum, most organizations have a designated data store/data warehouse containing extracted data from key sources and a frontend BI tool or reporting engine. A more extensive footprint might also include Big Data, a variety of ETL, data movement, and event processing solutions (such as Oracle Event Processing, or OEP), data quality and master data management solutions, a mixture of reporting, dashboard, and ad hoc query tools, advanced analytics tools for statistical analysis and data mining, and a real-time recommendation engine.

A complex analytics infrastructure that contains Oracle components might look like the following diagram in Figure 10-6.

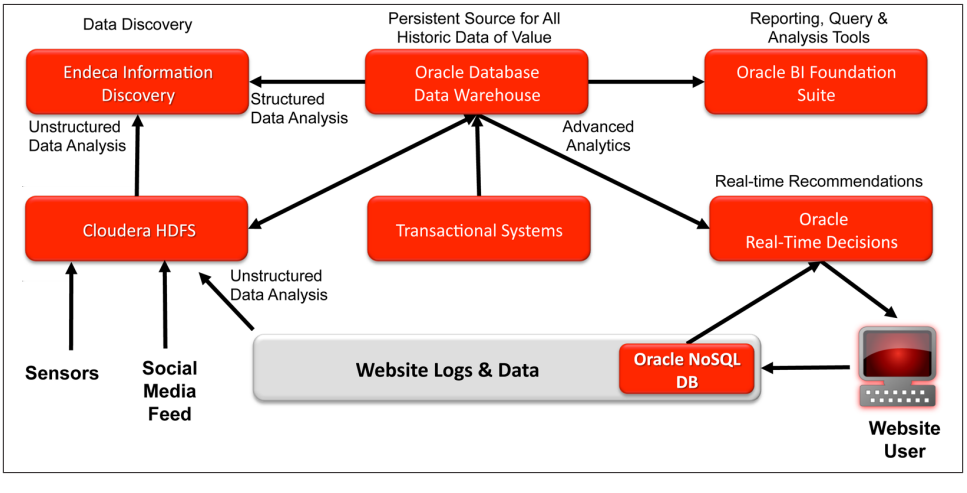


Figure 10-6. Oracle Analytics Infrastructure Footprint

In this scenario, we find structured data in the Oracle data warehouse gathered from transactional data sources. A variety of other data sources, including sensor data, social media feeds, and web log information, are analyzed in a Hadoop cluster. We use a data discovery tool, Endeca, to explore all of our data, structured and unstructured. We populate the data warehouse with MapReduce output containing the data of value we

find value in our Hadoop cluster. The broad business analyst community gains access to all of this data through BI tools using the Oracle BI Foundation Suite to access the data warehouse. Models of behavior are updated and sent to our real-time recommendation engine, Oracle Real-Time Decisions (RTD). RTD provides individual recommendations based on user profiles and user location as the website users traverse the site.

Best Practices

Those experienced in business intelligence generally agree that the following are typical reasons why these projects fail:

Failure to involve business users, IT representatives, sponsoring executives, and anyone else with a vested interest throughout the project process
: Not only do all of these groups provide valuable input for creating a business intelligence solution, but lack of support by any of them can cause a project to fail.

Overlooking the key reasons for the business intelligence infrastructure
: During the planning stages, IT architects can lose sight of the forces driving the creation of the solution.

Overlooked details and incorrect assumptions
: A less-than-rigorous examination of the environment can doom the project to failure.

Unrealistic time frames and scope
: As with all projects, starting the creation of a business intelligence solution with too short a time frame and too aggressive a scope will force the team to cut corners, resulting in the mistakes previously mentioned.

Failure to manage expectations
: Data warehouses and business intelligence solutions, like all technologies, are not a panacea. You must make sure that all members of the team, as well as the eventual users of the solution, have an appropriate set of expectations. While setting these expectations, you must remember that these consumers are almost always business users, so you should make sure they understand the real-world implications of various IT decisions.

Tactical decision making at the expense of long-term strategy
: Although it may seem overly time-consuming at the start, you must keep in mind the long-term goals of your project, and your organization, throughout the design and implementation process. Failing to do so delays the onset of problems, but it also increases the likelihood and severity of those problems.

Failure to leverage the experience of others
: There's nothing like learning from those who have succeeded on similar projects. It's almost as good to gain from the experience of others who have failed at similar tasks; at least you can avoid the mistakes that led to their failures.

Successful business intelligence projects require the continuous involvement of business analysts and users, sponsoring executives, and IT. Ignoring this often-repeated piece of advice is probably the single biggest cause of many of the most spectacular failures. Establishing this infrastructure has to produce a clear business benefit and an identifiable return on investment (ROI). Executive involvement is important throughout the process because business intelligence coordination often crosses departmental boundaries, and continued funding likely will come from high levels.

Your business intelligence project should provide answers to business problems that are linked to key business initiatives. Ruthlessly eliminate any developments that take projects in another direction. The motivation behind the technology implementation schedule should be the desire to answer critical business questions. Positive ROI from the project should be demonstrated during the incremental building process.

Common Misconceptions

Having too simplistic a view during any part of the building process (a view that overlooks details) can lead to many problems. Here are just a few of the typical (and usually incorrect) assumptions people make in the process of implementing a business intelligence solution:

- Sources of data are clean and consistent.
- Someone in the organization understands what is in the source databases, the quality of the data, and where to find items of business interest.
- Extractions from operational sources can be built and discarded as needed, with no records left behind.
- Summary data is going to be adequate, and detailed data can be left out.
- IT has all the skills available to manage and develop all the necessary extraction routines, tune the database(s), maintain the systems and the network, and perform backups and recoveries in a reasonable time frame.
- Development is possible without continuous feedback and periodic prototyping involving analysts and possibly sponsoring executives.
- The warehouse won't change over time, so "versioning" won't be an issue.
- Analysts will have all the skills needed to make full use of the infrastructure or the business intelligence tools.
- IT can control what tools the analysts select and use.

- Big Data will continue to exist in a silo separate from the rest of the analytics infrastructure.
- The number of users is known and predictable.
- The kinds of queries are known and predictable.
- Computer hardware is infinitely scalable, regardless of choices made.
- If a business area builds a data mart or deploys an appliance independently, IT won't be asked to support it later.
- Consultants will be readily available in a pinch to solve last-minute problems.
- Metadata or master data is not important, and planning for it can be delayed.

Effective Strategy

Most software and implementation projects have difficulty meeting schedules. Because of the complexity in business intelligence projects, they frequently take much longer than the initial schedule, and that is exactly what executives who need the information to make vital strategic decisions don't want to hear! If you build in increments, implementing working prototypes along the way, the project can begin showing positive return on investment, and changes in the subsequent schedule can be linked back to real business requirements, not just back to technical issues (which executives don't ordinarily understand).

You must avoid scope creep and expectations throughout the project. When you receive recommended changes or additions from the business side, you must confirm that these changes provide an adequate return on investment or you will find yourself working long and hard on facets of the infrastructure without any real payoff. The business reasoning must be part of the prioritization process; you must understand why trade-offs are made. If you run into departmental "turf wars" over the ownership of data, you'll need to involve key executives for mediation and guidance.

The pressure of limited time and skills and immediate business needs sometimes leads to making tactical decisions in establishing a data warehouse at the expense of a long-term strategy. In spite of the pressures, you should create a long-term strategy at the beginning of the project and stick to it, or at least be aware of the consequences of modifying it. There should be just enough detail to prevent wasted efforts along the way, and the strategy should be flexible enough to take into account business acquisitions, mergers, and so on.

Your long-term strategy must embrace emerging trends, such as the need to meet compliance initiatives or the need for highly available solutions. The rate of change and the volume of products being introduced sometimes make it difficult to sort through what is real and what is hype. Most companies struggle with keeping up with the knowledge

curve. Traditional sources of information include vendors, consultants, and data-processing industry consultants, each of whom usually has a vested interest in selling something. The vendors want to sell products; the consultants want to sell skills they have "on the bench," and IT industry analysts may be reselling their favorable reviews of vendors and consultants to those same vendors and consultants. Any single source can lead to wrong conclusions, but by talking to multiple sources, some consensus should emerge and provide answers to your questions.

The best place to gain insight is discussing business intelligence projects with other similar companies—at least at the working-prototype stage—at conferences. Finding workable solutions and establishing a set of contacts to network with in the future can make attendance at these conferences well worth the price—and can be more valuable than the topics presented in the standard sessions.

CHAPTER 11
Oracle and High Availability

The data stored in your Oracle databases is one of your organization's most valuable assets. Protecting and providing timely access to this data when it is needed for business decisions is crucial for any Oracle site.

As an Oracle database administrator, system administrator, or system architect, you'll probably use a variety of techniques to ensure that your data is adequately protected from catastrophe. Of course, implementing proper backup operations is the foundation of any availability strategy, but there are other ways to avoid a variety of possible outages that could range from simple disk failures to a complete failure of your primary site. In addition, there are software solutions that can help avoid the loss of availability that these failures can create.

Computer hardware is, by and large, extremely reliable, and that can tempt you to postpone thinking about disaster recovery and high availability. Most software is also very reliable, and the Oracle database protects the integrity of the data it holds even in the event of software failure. However, hardware and software will fail occasionally. The more complicated the infrastructure, the greater the likelihood of downtime at the worst time.

The difference between inconvenience and disaster is often the presence or absence of adequate recovery plans and options. This chapter should help you understand all of the options available when deploying Oracle so you can choose the best approach for your site.

With Oracle, you can guarantee that your precious data is highly available by leveraging built-in capabilities such as instance recovery or options such as Active Data Guard and Real Application Clusters. However, equally important in deploying a high-availability solution is the implementation of the appropriate procedures to safeguard your data. This chapter covers these various aspects of high availability in what Oracle often describes as a Maximum Availability Architecture (MAA).

What Is High Availability?

Before we can begin a discussion of how to ensure a high level or maximum level of availability, you need to understand the exact meaning of the term *availability*.

Availability can mean different things for different organizations. For this discussion, we'll consider a system to be available when it is *up* (meaning that the Oracle database can be accessed by users) and *working* (meaning that the database is delivering the expected functionality to business users at the expected performance). In other words, the system is exceeding service level agreements (SLAs).

When the database is unavailable, we refer to that as *downtime*. Of course, the goal is to eliminate unplanned downtime caused by server, storage, network, and/or software failures, and human error, a common source of problems. You will also want to minimize planned downtime during system and Oracle database changes, structural data changes, and application changes.

Most businesses depend on data being available when needed to make important decisions. Accessibility to data via web-based solutions means that database failures can have a broad impact. Failures of systems accessed by a wider community outside of company boundaries are, unfortunately, immediately and widely visible and can seriously impact a company's financial health and image. Consider a web-based customer service application provided by a package shipping company that enables customers to perform package tracking. As these customers come to depend on such service, any interruption in that service could cause these same customers to move to competitors.

Taking this a step further, consider complexities in accessing data that resides in multiple systems. Integrating multiple systems can increase chances of failures and could cause access to an entire supply chain to be unavailable.

To implement Oracle databases that are highly available, you must design an infrastructure that can mitigate unplanned and planned downtime. You must embrace techniques that enable rapid recovery from disasters, such as implementing appropriate backup solutions. The goal of all of this is to ensure business continuity during situations that range from routine maintenance to dire circumstances. In some organizations, the requirements are 24/7/365. In others, less stringent business requirements might drive different solutions.

Measuring High Availability

To provide some perspective on what might be described as high availability, consider Table 11-1, which translates the percentage of availability into days, minutes, and hours of annual downtime based on a 365-day year.

Table 11-1. Percent availability and downtime per year

% availability	Downtime per year		
	Days	Hours	Minutes
95.000	18	6	0
96.000	14	14	24
97.000	10	22	48
98.000	7	7	12
99.000	3	15	36
99.500	1	19	48
99.900	0	8	46
99.990	0	0	53
99.999	0	0	5

Large-scale systems that achieve over 99 percent availability can cost hundreds of thousands of dollars or more to design and implement and can have high ongoing operational costs. Marginal increases in availability can require large incremental investments in system components. While moving from 95 to 99 percent availability is likely to be costly, moving from 99 to 99.999 percent is usually costlier still.

You might also consider *when* the system must be available. A required availability of 99 percent of the time during normal working hours (e.g., from 8 a.m. to 5 p.m.) is very different from 99 percent availability based on a 24-hour day. In the same way that you must carefully define your required levels of availability, you must also consider the hours during which availability is measured. For example, many companies take orders during "normal" business hours. The cost of an unavailable order-entry system is very high during the business day, but drops significantly after hours. Thus, planned downtime can make sense after hours, which will, in turn, help reduce unplanned failures during business hours. Of course, in some multinational companies, the global reach implies that the business day never ends.

A requirement that a database be available 24/7/365 must be put in the context of the cost in deploying and maintaining such a solution. An examination of the complexity and cost of very high levels of availability will sometimes lead to compromises that reduce requirements for this level of availability.

The costs of achieving high availability are certainly justified in many cases. It might cost a brokerage house millions of dollars for each hour that key systems are down. A less-demanding use case, such as human resources reporting, might be perfectly fine with a lesser level of availability. But, regardless of the cost of lost business opportunity, an unexpected loss of availability can cut into the productivity of the lines of business and IT staff alike.

The System Stack and Availability

There are many different causes of unplanned downtime. You can prevent some very easily, while others require significant investments in site infrastructure, including software, servers, storage, networks, and appropriate employee skills. Figure 11-1 lists some of the most frequent causes of unplanned downtime.

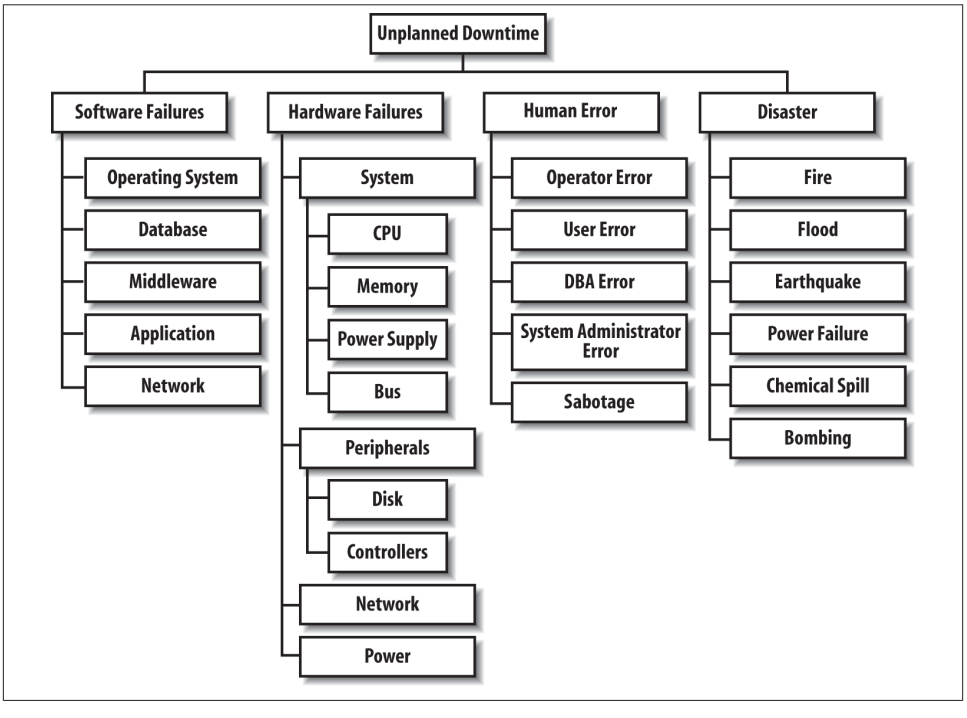


Figure 11-1. Causes of unplanned downtime

When creating a plan to guarantee the availability of an application and Oracle Database, you should consider all of the situations shown in Figure 11-1 as well as other potential causes of system interruption that are specific to your own circumstances. As with all planning, it's much better to consider all possible situations, even if you quickly dismiss them, than to be caught off guard when an unexpected event occurs.

A complete system is composed of hardware, software, and networking components operating as a technology *stack*. Ensuring the availability of individual components doesn't necessarily guarantee system availability. Different strategies and solutions exist for achieving high availability for each of the system components. Figure 11-2 illustrates the technology stack used to deliver a potential system.

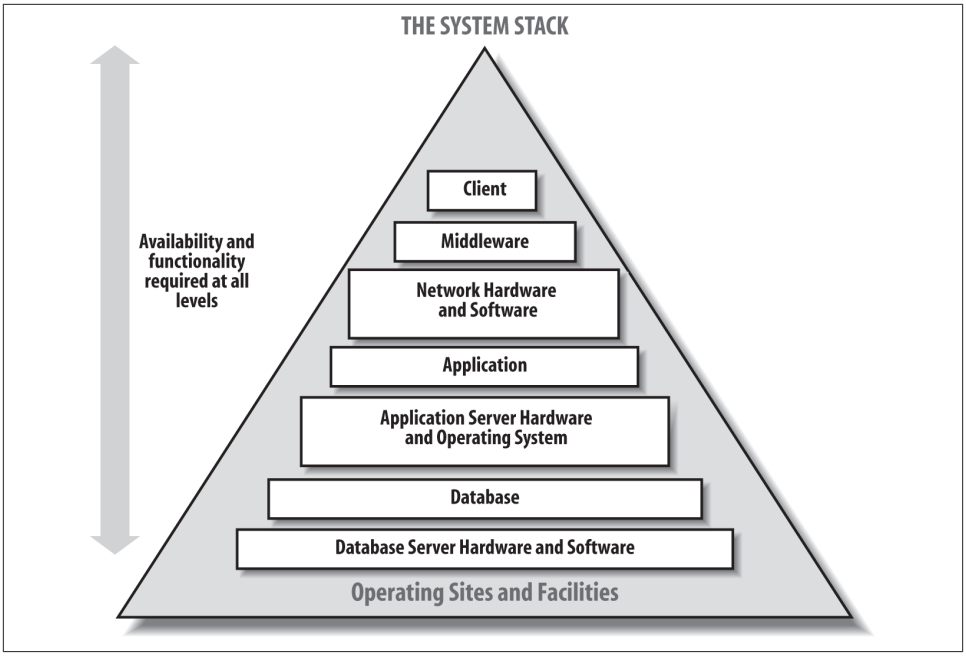


Figure 11-2. Technology components in the system stack

As this figure shows, a variety of physical and logical layers must cooperate to deliver an application. Failures in the components above the Oracle database can effectively prevent access to the database even though the database itself may be available. The server hardware, software, and database itself serve as the foundation for the stack. When an Oracle database fails, it immediately affects the higher levels of the stack. If the failure results in lost or corrupted data, the overall integrity of the application may be affected.

The potential threats to availability span all of the components involved in an application system, but in this chapter we'll examine only availability issues relating specifically to the database.

Server Hardware, Storage, and Database Instance Failure

The failure of servers or storage hosting a database can be one of the most abrupt causes of unplanned downtime. A server may crash because of hardware problems, such as the failure of a power supply, or because of software problems, such as a process that begins to consume all the machine's CPU resources. Even if the underlying server platform is fine, the Oracle instance itself can fail. Whatever the cause of the crash, the effect on Oracle is the same—the instance cannot deliver its promised functionality. Remember that when an Oracle Database crashes, it is the instance that crashes, not the database

(as described in Chapter 2). Even if the system fails, the failure will not imperil any data that's already safely stored within the disk files used by the Oracle database.

The impact of a crash will depend on the activity in progress at the time of the crash. Any connected sessions will no longer have a server process to which to talk. All active queries and transactions will be abruptly terminated. The process of cleaning up the resulting mess is called *instance recovery* or *crash recovery*.

Telltale Error Messages

The following two error messages are often good indicators that an Oracle instance is down:

```
ORA-03113: End-of-file on communication channel
```

This message is usually received by clients that try to resubmit an operation that failed due to an instance failure. The message is somewhat cryptic but becomes clear if you interpret it very literally. Oracle works using a pipe to communicate between the client application and its associated server process in the Oracle instance. When the instance fails, the client's server process ceases to exist, so there is no one listening on the other end of the pipe. The communication channel between the client and the server is no longer valid.

```
ORA-01034: Oracle not available
```

This terse message means that when the client requested a connection to the Oracle instance, the instance was not there. Clients that try to connect to a failed instance will typically get this message. The client can connect to the Listener, but when the Listener attempts to hand the client off to the requested Oracle instance, the `ORA-01034` condition results.

What Is Instance Recovery?

When you restart an Oracle instance after a failure, Oracle detects that a crash occurred using information in the control file and the headers of the database files. Oracle then performs instance recovery automatically and uses the online redo logs to guarantee that the physical database is restored to a consistent state as it existed at the time of the crash. This includes two actions:

- All committed transactions will be recovered.
- In-flight transactions will be rolled back or undone.

Note that an in-flight transaction might be one that a user didn't commit or one that was committed by the user but not confirmed by Oracle before the system failure. A transaction isn't considered committed until Oracle has written the relevant details of

the transaction to the current online redo log and has sent back a message to the client application confirming the committed transaction.

Phases of Instance Recovery

Instance recovery has two phases: roll forward and roll back.

Recovering an instance requires the use of the redo logs, described in Chapter 2. The redo logs contain a recording of all the physical changes made to the database as a result of transactional activity, both committed and uncommitted.

The checkpoint concept, also described in Chapter 2, is critical to understanding crash recovery. When a transaction is committed, Oracle writes all associated database block changes to the current online redo log. The actual database blocks may have already been flushed to disk, or may be flushed at some later point. This means that the online redo log can contain changes not yet reflected in the actual database blocks stored in the datafiles. Oracle periodically ensures that the data blocks in the datafiles on disk are synchronized with the redo log to reflect all the committed changes up to a point in time. Oracle does this by writing all the database blocks changed by those committed transactions to the datafiles on disk. This operation is called a *checkpoint*. Completed checkpoints are recorded in the control file, datafile headers, and redo log.

The DBA defines bounded database startup recovery times through Enterprise Manager. Behind the scenes, self-tuned checkpoint processing (provided by Oracle since Oracle Database 10*g*) accounts for the increase in I/O that occurs as the database writer flushes all database blocks to disk to bring datafiles up to the time of the checkpoint. In older versions of the Oracle database, DBAs would sometimes reduce the checkpoint interval or timeout and reduce amounts of data between checkpoints for faster recovery times, but would also introduce the overhead of more frequent checkpoints and their associated disk activity. A common strategy to minimize the number of checkpoints was to set the initialization file parameters so that checkpoints would occur only with log switches, a situation that would normally occur anyway.

At any point in time, the online redo logs will be ahead of the datafiles by a certain amount of time or number of committed transactions. Instance recovery closes this gap and ensures that the datafiles reflect all committed transactions up to the time the instance crashed. Oracle performs instance recovery by rolling forward through the online redo log and replaying all the changes from the last completed checkpoint to the time of instance failure. This operation is called the *roll forward* phase of instance recovery and is now sometimes also referred to as *cache recovery*.

While implementing roll forward recovery, Oracle reads the necessary database blocks into the System Global Area and reproduces the changes that were originally applied to the data blocks, including reproducing the undo or rollback information. UNDO segments are composed of extents and data blocks just like tables, and all changes to UNDO

segment blocks are part of the redo for a given transaction. For example, suppose that a user changed an employee name from "John" to "Jonathan." As Oracle applies the redo log, it will read the block containing the employee row into the cache and redo the name change. As part of recovering the transaction, Oracle will also write the old name "John" to an UNDO segment, as was done for the original transaction.

When the roll forward phase is finished, all the changes for committed and uncommitted transactions have been reproduced. The uncommitted transactions are in-flight once again, just as they were at the time the crash occurred. These in-flight transactions must be rolled back to return to a consistent state. This next phase is called *rollback* or *transaction recovery*.

Oracle opens the database after the roll forward phase of recovery and performs the rollback of uncommitted transactions in the background in what is called *deferred rollback*. This process reduces database downtime and helps to reduce the variability of recovery times. If a user's transaction begins working in a database block that contains some changes left behind by an uncommitted transaction, the user's transaction will trigger a foreground rollback to undo the changes and will then proceed when rollback is complete. This action is transparent to the user—they don't receive error messages or have to resubmit the transaction.

Protecting Against System Failure

There are a variety of approaches you can take to help protect your system against the ill effects of system crashes and other failures, including the following:

- Providing component redundancy
- Deploying Data Guard to provide an alternate site in case of primary site failure
- Deploying Real Application Clusters for database continuity in the event of failure of an instance
- Deploying application continuity or Transparent Application Failover software services

Component Redundancy

As basic protection, the various hardware components that make up the Oracle Database server itself must be fault-tolerant. *Fault-tolerance*, as the name implies, allows the overall hardware system to continue to operate even if one of its components fails. This, in turn, implies redundant components and the ability to detect component failure and seamlessly integrate the failed component's replacement. The major system components that should have redundancy include the following:

- Disk drives
- Disk controllers
- Flash memory
- CPUs
- Power supplies
- Cooling fans
- Network cards

Systems designed with this in mind include Oracle's engineered systems. For example, the Oracle Exadata Database Machine and other Oracle engineered systems designed to run the Oracle Database are pre-configured with redundant components to eliminate single point of failure for any of the above.

Disk failure is the largest area of exposure for hardware since disks have the shortest mean times to failure of any of the components in a computer system. Disks also present the greatest variety of redundant solutions, so discussing that type of failure here should provide a good example of how high availability can be implemented with hardware.

Disk Redundancy

Although the mean time to failure of an individual disk drive is very high, the ever-increasing number of disks used for today's very large databases results in more frequent failures. Protection from disk failure is usually accomplished using RAID (Redundant Array of Inexpensive Disks) concepts. The use of redundant storage has become common for systems of all sizes and types for two primary reasons: the real threat of disk failure and the proliferation of packaged, relatively affordable RAID solutions.

RAID uses one of two concepts to achieve redundancy:

Mirroring
: The actual data is duplicated on another disk in the system.

Striping with parity
: Data is striped on multiple disks, but instead of duplicating the data itself for redundancy, a mathematical calculation termed *parity* is performed on the data and the result is stored on another disk. You can think of parity as the sum of the striped data. If one of the disks is lost, you can reconstruct the data on that disk using the surviving disks and the parity data. The lost data represents the only unknown variable in the equation and can be derived. You can conceptualize this as a simple formula:

```
A + B + C + D = E
```

in which A–D are data striped across four disks and E is the parity data on a fifth disk. If you lose any of the disks, you can solve the equation to identify the missing component. For example, if you lose the B drive you can solve the formula as:

```
B = E - A - C - D
```

There are a number of different disk configurations or types of RAID technology, which are formally termed *levels*. Table 11-2 summarizes the most relevant levels of RAID in a bit more detail, in terms of their cost, high availability, and the way Oracle uses each RAID level.

Table 11-2. RAID levels relevant to high availability

Level	Disk configuration	Cost	Comments	Oracle usage
0	Simple striping, no redundancy	Same cost as unprotected storage.	The term RAID-0 is used to describe striping, which increases read and write throughput. However, this is not really RAID, as there is no actual redundancy.	Striping simplifies administration for Oracle datafiles. Suitable for all types of data for which redundancy isn't required.
1	Mirroring	Twice the cost of unprotected storage.	Same write performance as a single disk. Read performance may improve through servicing reads from both copies.	Lack of striping adds complexity of managing a larger number of devices for Oracle. Often used for redo logs, since the I/O for redo is typically relatively small sequential writes. Striped arrays are more suited to large I/Os or to multiple smaller, random I/Os.
0+1	Striping and mirroring	Twice the cost of unprotected storage.	Best of both worlds—striping increases read and write performance and mirroring for redundancy avoids "read-modify-write" overhead of RAID-5.	Same usage as RAID-0, but provides protection from disk failure.
5	Striping with rotating or distributed parity	Storage capacity is reduced by 1/*N*, where *N* is the number of disks in the array. For example, the storage is reduced by 20%, or 1/5 of the total disk storage, for a 5-disk array.	Parity data is spread across all disks, avoiding the potential bottleneck found in some other types of RAID arrays. Striping increases read performance. Maintaining parity data adds additional I/O, decreasing write performance. For each write, the associated parity data must be read, modified, and written back to disk. This is referred to as the "read-modify-write" penalty.	Cost-effective solution for all Oracle data except redo logs. Degraded write performance must be taken into account. Popular for reads where adequate I/O is provided. Write penalties may slow loads and index builds. Often avoided for high-volume OLTP due to write penalties. Some storage vendors, such as EMC, have proprietary solutions (RAID-S) to minimize parity overhead on writes.

Figure 11-3 illustrates the disk configurations for various RAID levels.

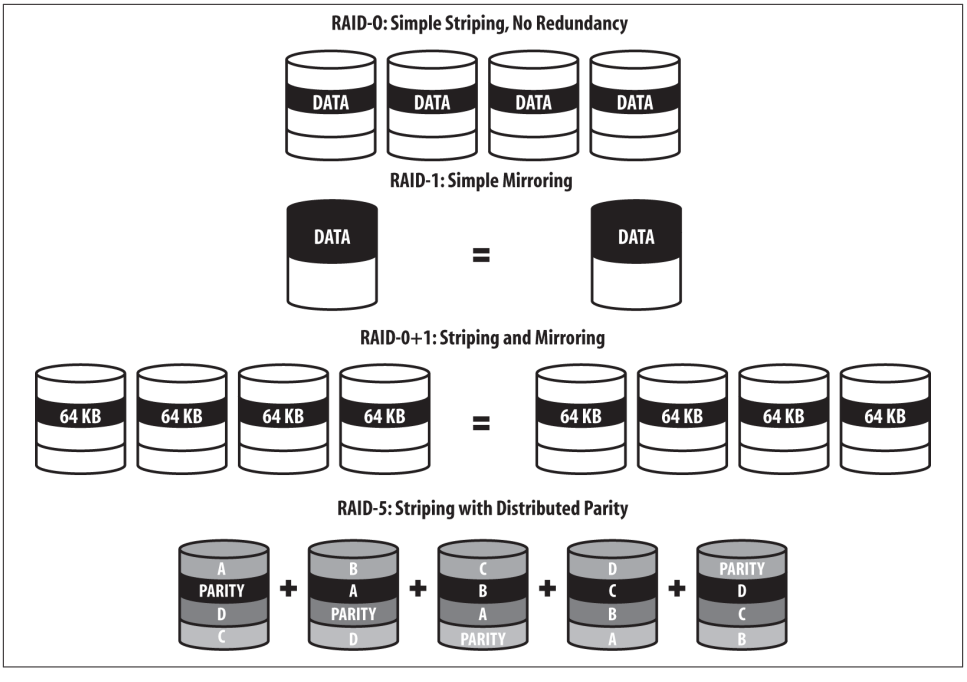


Figure 11-3. RAID levels commonly used with an Oracle Database

Automatic Storage Management

Oracle Database versions dating back to Oracle Database 10*g* include Automatic Storage Management (ASM). We introduced ASM in Chapter 5 and described its manageability considerations. ASM enables you to create a pool of storage on disk groups and then manages the placement of database files on the storage. ASM features enable it to replace non-Oracle file systems and logical volume managers for files managed by the Oracle Database. An ASM instance manages each of the disks in the disk group, and one ASM instance is provided for each database node in a RAC environment.

ASM is an Oracle-recommended MAA solution to handle storage failures and data corruption. It provides "Striping and Mirroring Everything" (SAME) for many types of disks, including "Just a Bunch of Disks" (JBOD) arrays. You can specify groups of disks, and designate a failure group to be used in the result of a disk failure. Mirroring can also be set up on a per-file basis, and you can specify one mirror (normal redundancy) or two mirrors (high redundancy). ASM includes the ability to detect disk "hot spots" and redistribute data to avoid disk bottlenecks, as well as the capability of adding disks to a disk group without any interruption in service. DBAs add the disks to disk groups or remove disks from disk groups using Oracle Enterprise Manager.

Stored data is automatically rebalanced when disks are added or removed. When a drive fails, remirroring to remaining drives is automatic. These features make ASM ideal for managing a database storage grid and allow you to use cheaper disk systems while obtaining higher levels of availability and performance. Fast mirror resynchronization, supported since Oracle Database 11*g*, enables faster recovery from transient failures since ASM will only resynchronize changed ASM disk extents for limited duration failures.

Oracle Database 12*c* introduces *Flex ASM* enabling ASM servers to be run on a separate physical server from the Oracle Database servers. Large clusters of ASM instances can be configured to support large numbers of database clients while reducing the overall ASM footprint that would be present if Flex ASM were not deployed.

ASM and RAID Levels

Given the popular adoption of ASM and SAME for managing Oracle Database storage on Oracle's engineered systems such as the Oracle Exadata Database Machine and other Oracle and non-Oracle server and storage platforms, RAID 0+1 has become the most common choice for deployment. For example, when Oracle installs the Exadata Database Machine, it includes full mirroring and striping as part of the standard installation. Advanced compression techniques such as Hybrid Columnar Compression more than offset additional storage required to deploy in this configuration.

Site and Computer Server Failover

The Oracle Database will recover automatically from a system crash. Automatic recovery protects data integrity, critical in a relational database, but also causes some downtime as the Oracle Database recovers. When a hardware failure occurs, the ability to quickly detect a system crash and initiate recovery is crucial to minimizing the associated downtime.

When an individual server fails, the instance(s) running on that server fail as well. Depending on the cause, the failed server may not return to service quickly or be noticed immediately. Either way, companies that wish to protect their systems from the failure of a server typically employ multiple servers to achieve *failover*. Although failover doesn't directly address the issue of the reliability of the underlying hardware, automated failover reduces downtime from hardware failure. The primary means to mitigate unplanned downtime caused by server failures is through the use of Data Guard or Real Application Clusters (RAC). Data Guard is also useful for protecting against site failures while RAC is noteworthy for also providing timely database instance failure recovery. We'll cover each of these in subsequent sections.

Protection from the complete failure of your primary Oracle data center site can pose significant challenges. Your organization must carefully evaluate the risks to its primary site. These risks include physical and environmental problems as well as hardware risks. For example, is the data center in an area prone to floods, tornadoes, or earthquakes? Are power failures a frequent occurrence?

Protection from primary site failure involves monitoring of and redundancy controls for the following:

- Data center power supply
- Data center climate control facilities
- Database server redundancy
- Database redundancy
- Data redundancy

The first two items on the list are aimed at preventing the failure of the data center. Data server redundancy provides protection from node failure within a data center but not from complete data center loss. Should the data center fail completely, the last two items —database redundancy and data redundancy—enable disaster recovery. In reality, this requires data center redundancy if the goal is to run at previous production levels of performance after the disaster.

Oracle Data Guard and Site Failures

Data Guard is an Oracle-recommended MAA solution that is used to recover databases from site and storage failures as well as data corruption. It is included with the Enterprise Edition of Oracle and supports database configurations for physical standby (an MAA best practice), snapshot standby for testing, and logical standby to reduce planned downtime. Applying redo to up to 30 physical standby Oracle databases is possible from the primary Oracle database cascading via a remote location onto the standbys. As of Oracle Database 12*c*, Data Guard can also be used for disaster recovery of multitenant container databases (CDBs).

The concept of a physical standby database is simple—keep a copy of the database files at a second location, ship the redo logs to the second site, and apply them to the copy of the Oracle database. This process can keep the standby database "a few steps" behind the primary database when deployed asynchronously. If the primary site fails, a standby database is opened and becomes the production database. The potential data loss is limited to the transactions in any redo logs that have not been shipped to the standby site. Figure 11-4 illustrates a standby Oracle Database.

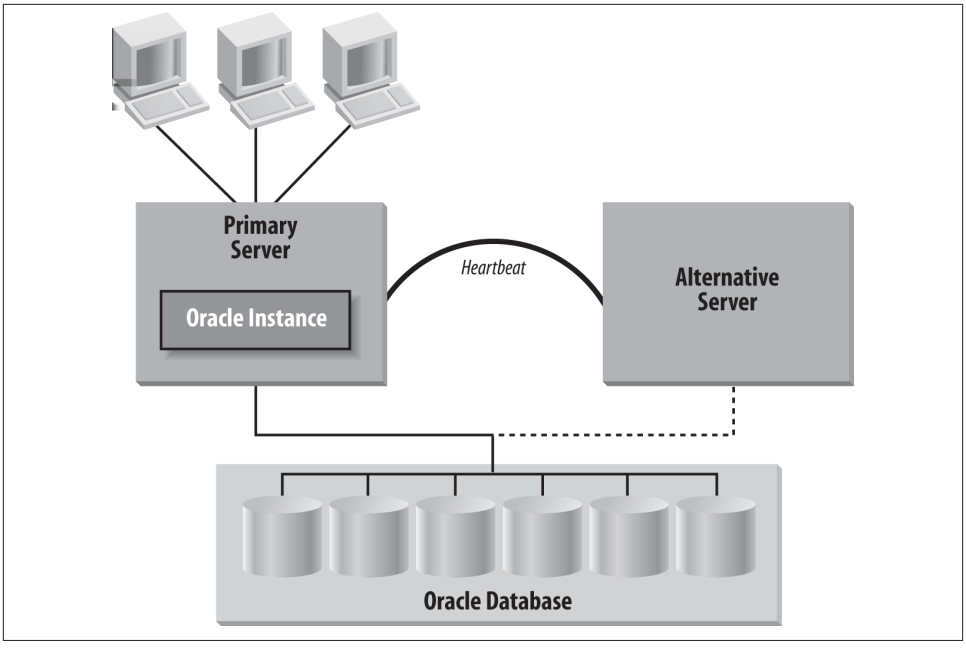


Figure 11-4. Standby Oracle Database

There are three possible causes of lost data in the event of primary site failure when deploying a physical standby database:

- Archived redo logs have not been shipped to the standby site.
- Filled online redo logs have not been archived yet.
- The current online redo log is not a candidate for archiving until a log switch occurs.

Archiving of redo logs to a destination on the primary server as well as on multiple remote servers serving as standby sites is automated and Oracle applies the archived redo logs to the standby database as they arrive. A fast-start failover is supported so redo loss exposure does not exceed the limits the administrator sets.

The Oracle Data Guard broker enables monitoring and control of physical and logical standby databases and components. A single command can be used to perform failover. Oracle Database 12*c* added configuration health checks, resumable switchover operations, streamlined role transitions, support for the cascaded standby configurations, and user configurable thresholds regarding SLAs for recovery to the supported features. Oracle Enterprise Manager provides a Data Guard Manager GUI for setting up, monitoring, and managing the standby database.

Oracle Active Data Guard and Zero Data Loss

Active Data Guard is an option to Oracle Database Enterprise Edition and deployed where data loss must be completely avoided. It features a real-time apply, in which redo data is applied at the standby Oracle Database as soon as it is received, and automatic block repair. The DBA has a choice of three different data loss modes that can be specified. Maximum protection mode guarantees that switching over to the standby database will not result in any lost data as each log write must also be completed to a remote logfile before transactions commit on the primary database and, in the event that this does not occur, the primary database is shut down. Maximum performance mode provides a high level of protection but will not impact the performance of the primary database, as transactions commit as soon as they are written to the online redo log and then written asynchronously to one or more standby databases. Maximum availability mode ensures that transactions don't commit unless they are written to the online redo log and one or more standby databases, but will fall back to maximum performance mode if writing to a standby is not possible.

Active Data Guard also enables the standby Oracle database to be used for reporting and, since the Oracle Database 12*c* release, you can also query the standby database with DML to global temporary tables while the redo apply is active. The ability to offload reporting requests provides flexibility for reporting and queries and can help performance on the primary server while making use of the standby server. Active Data Guard for Oracle Database12*c* also supports scaling of reads across multiple standby databases simultaneously.

The Far Synch capabilities introduced with Active Data Guard for Oracle Database12*c* enable zero-loss primary and standby configurations that are separated by distances that can span continents. A lightweight database instance consisting of a control file and archive logfile but no recovery or datafiles is deployed to a system outside of the primary database. Redo is transported to this location synchronously, and then is forwarded asynchronously to the distant standby database, the failover target. If the primary database fails, failover is executed as usual to the failover with zero data loss.

Oracle GoldenGate and Replication

An alternative for guarding against site and computer failures that cause unplanned downtime is to implement client and application failover leveraging log-based change data capture and replication software, such as Oracle's GoldenGate. These types of solutions are typically used where there are heterogeneous server platforms and/or heterogeneous database implementations.

This type of platform failover architecture requires no additional Oracle software beyond GoldenGate. However, a solution of this type requires a great deal of additional scripting and testing. For example, data that is in a queue to be replicated from a server that goes down will not be available at the secondary site where GoldenGate replicates

to. Maintaining consistency of data across the two servers when the first is brought back online also requires very careful scripting.

The extent of the data divergence and potential data loss resulting from the divergence is a very important consideration since transactions performed at the primary site are replicated some time later to the secondary site. Synchronous replication (e.g., Active Data Guard) is used when there is no tolerance for data divergence or lost data such that the data at the secondary site *must* match the primary site at all times and reflect all committed transactions.

GoldenGate provides asynchronous replication, so until the deferred transaction queue is "pushed" to the secondary site replicating the changes, the data at the secondary site will differ from the primary site data. If the primary database is irrevocably lost, any transactions that had not been pushed from the deferred queue will also be lost. Typically, asynchronous replication adds less overhead than synchronous replication, since the replication of changes can be efficiently batched to the secondary site.

Figure 11-5 illustrates synchronous and asynchronous replication.

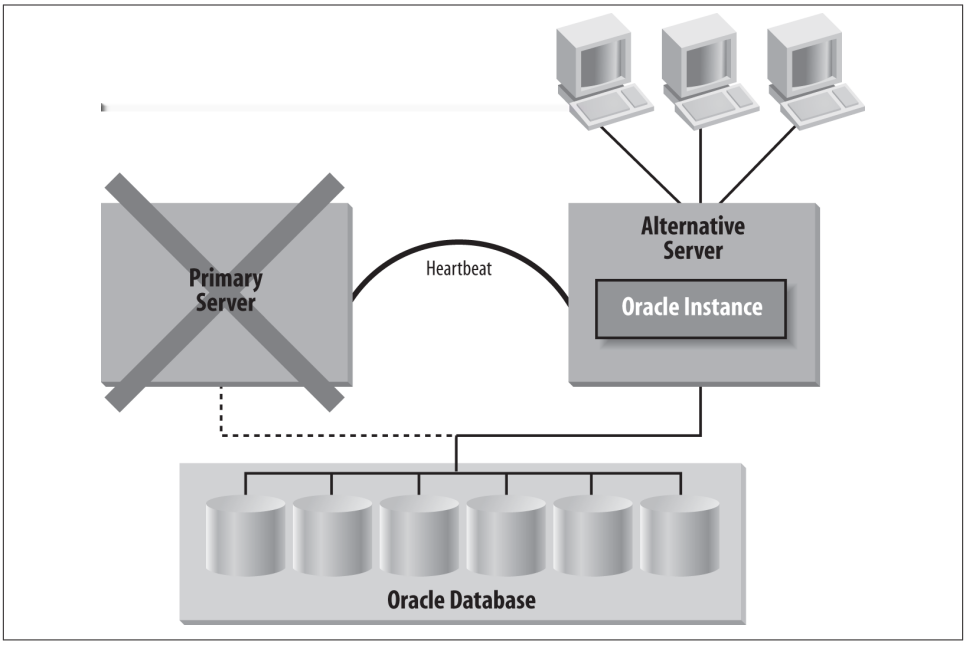


Figure 11-5. Oracle replication for redundant data

Considerations in setting up an asynchronous replication environment include the following:

Tolerance for data divergence
: The smaller the data divergence, the more individual replication actions will have to be performed. You will reduce the resources needed to implement the replication by increasing the data divergence.

Performance requirements
: Since replication requires resources, frequent replication can have an impact on performance.

Network bandwidth
: Since replication uses network bandwidth, you have to consider the availability of this resource.

Distance between sites
: The more distance between sites, the longer the physical transfer of data will take.

Site and network stability
: If a site or a network goes down, all replications that use that network or are destined for that site will not be received. When either of these resources comes back online, the stored replication traffic can have an impact on the amount of time it takes to recover the site.

Experience level of your database administrators
: Even the most effective replication plan can be undone by DBAs who aren't familiar with replication.

While offering more flexibility, the added complexity and opportunity for data inconsistencies generally cause organizations to avoid using asynchronous replication by itself as part of a solution for unplanned downtime today and so GoldenGate is not on the list of Oracle MAA recommendations. That said, GoldenGate is often used during planned downtime for migrations and upgrades, and sometimes used in combination with Active Data Guard or Real Application Clusters as part of an MAA strategy where multimaster replication and other advanced replication techniques are desired.

Real Application Clusters and Instance Failures

Real Application Clusters (RAC) provides an Oracle-recommended MAA solution for instance or computer failures. With RAC, the Oracle database is spread across multiple nodes and each node has an active Oracle instance. Clients can connect to any of the instances to access the same database.

Using Real Application Clusters to spread the workload over several nodes or servers will result in a lower percentage of each server's resources being used in normal operating conditions. Each node or server in the cluster must devote some overhead to maintaining its role in the cluster, although this overhead is minimal. You will have to weigh the benefits of carrying on with some performance degradation in the event of a node failure versus the cost of buying more nodes. The economics of your situation may dictate that a decrease in performance in the event of a node failure is more palatable than a larger initial outlay for more nodes or larger systems.

Because each Oracle instance runs on its own node, if a node fails, the instance on that node also fails. The overall Oracle database remains available since surviving instances are still running on other working nodes.

Figure 11-6 illustrates Real Application Clusters on a simple two-node cluster.

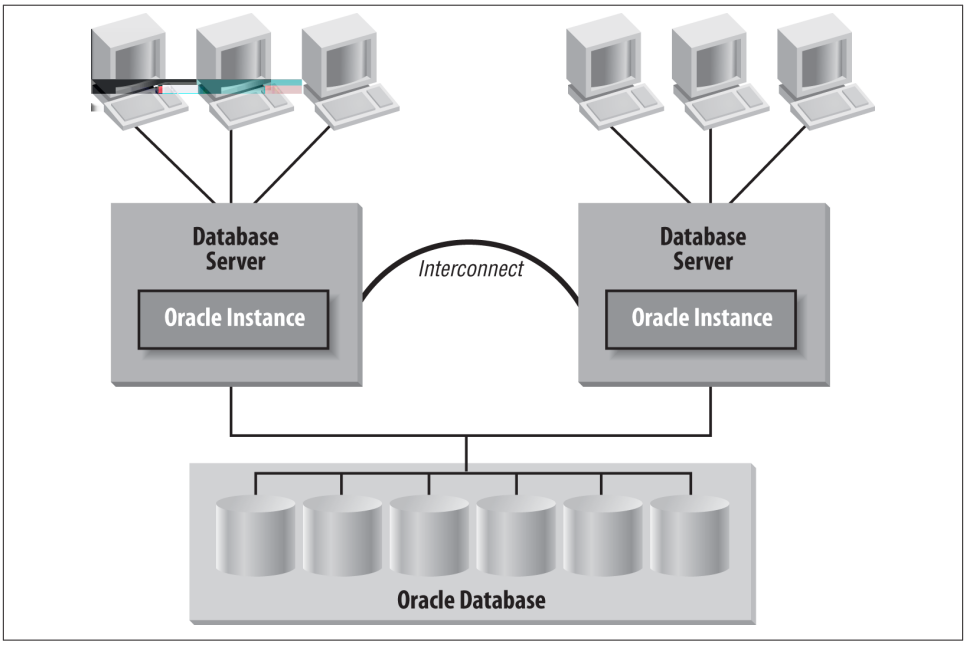


Figure 11-6. Oracle Real Application Clusters on a cluster

With Oracle Database 12*c*, it is possible to configure large RAC clusters as *Oracle Flex Clusters* that are deployed using a hub-and-spoke architecture. Hub nodes are tightly connected and have direct access to shared storage while anchoring Leaf nodes. The Leaf nodes are loosely connected and may or may not have direct access to shared storage. If you choose to install an Oracle Flex Cluster during the installation of Oracle Clusterware, Oracle Flex ASM is installed by default since it is required by a Flex Cluster.

Real Application Clusters increases availability by enabling avoidance of complete database blackouts. With simple hardware failover, the database is completely unavailable until node failover, instance startup, and crash recovery are complete. With RAC, clients can connect to a surviving instance any time. Clients may be able to continue working with no interruption, depending on whether the data they need to work on was under the control of the failed instance. You can think of the failure of a Real Application Clusters instance as a potential database "brownout," as opposed to the guaranteed blackout caused by hardware failover.

The multiple Oracle database instances provide protection for each other—if an instance fails, one of the surviving instances will detect the failure and automatically initiate RAC recovery. This type of recovery is different from the hardware failover discussed previously. No actual "failover" occurs—no disk takeover is required, since all nodes already have access to the disks used for the database. There is no need to start an Oracle instance on the surviving node or nodes, since Oracle is already running on all the nodes. The Oracle software performs the necessary actions without using scripts; the required steps are an integral part of Real Application Clusters software.

The phases of Real Application Clusters recovery are the following:

Cluster reorganization
: When an instance failure occurs, Real Application Clusters must first determine which nodes of the cluster remain in service. Each database group member votes on what members are part of the current group. Based on arbitration, a correct current group configuration is established. The time required for this operation is very brief.

Lock database rebuild
: The lock database, which contains the information used to coordinate Real Application Clusters traffic, is distributed across the multiple active instances. Therefore, a portion of that information is lost when a node fails. The remaining nodes have sufficient redundant data to reconstruct the lost information. Once the cluster membership is determined, the surviving instances reconstruct the lock database. The time for this phase depends on how many locks must be recovered, as well as whether the rebuild process involves a single surviving node or multiple surviving nodes. Oracle speeds the lock remastering process by optimizing lock master locations in the background while users are accessing the system.

Instance recovery
: Once the lock database is rebuilt, the redo logs from the failed instance perform crash recovery. This is similar to single-instance crash recovery—a roll forward phase followed by a nonblocking, deferred rollback phase. The key difference is that the recovery isn't performed by restarting a failed instance. Rather, it's performed by the instance that detected the failure.

As a clustered solution, RAC avoids the various activities involved in disk takeover: mounting volumes, validating filesystem integrity, opening Oracle database files, and so on. Not performing these activities significantly reduces the time required to achieve full system availability. RAC gained in popularity partly because it eliminated the creation and maintenance of the complex scripts typically used to control the activities for hardware failover. For example, there is no need to script which disk volumes will be taken over by a surviving node.

While RAC recovery is in progress, clients connected to surviving instances remain connected and can continue working. In some cases users may experience a slight delay in response times, but their sessions aren't terminated. Clients connected to the failed instance can reconnect to a surviving instance and can resume working. Uncommitted transactions will be rolled back and will have to be resubmitted. Queries that were active will also be terminated, but a seamless transition can be provided by Transparent Application Failover, as we'll describe in the next section.

Oracle Transparent Application Failover

Transparent Application Failover (TAF) provides seamless migration of users' sessions from one Oracle instance to another. You can use TAF to mask the failure of an instance for transparent high availability or for migration of users from a highly active instance to a less active one. Figure 11-7 illustrates TAF with Real Application Clusters.

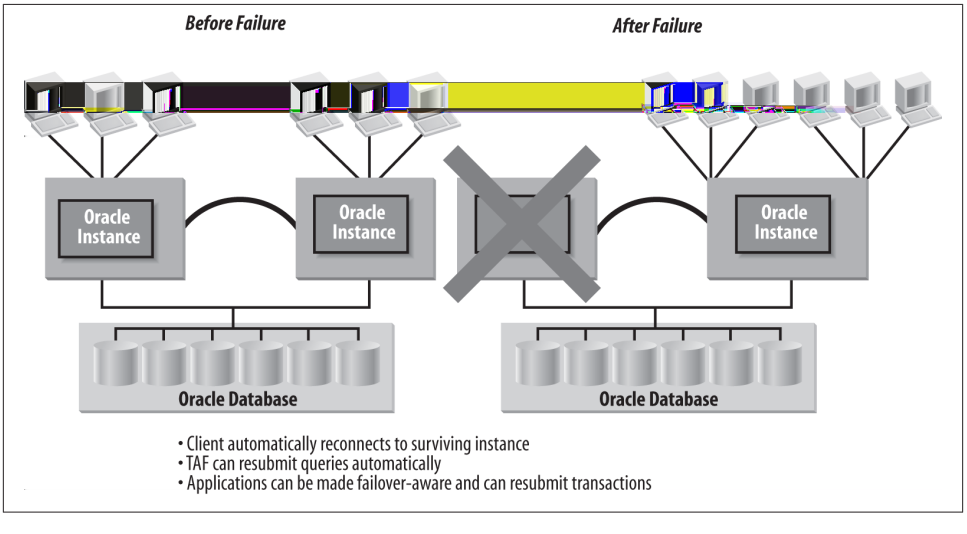


Figure 11-7. Failover with TAF and Real Application Clusters

As shown in this figure, TAF can automatically reconnect clients to another instance of the database, which provides access to the same Oracle database as the original instance. The high availability benefits of TAF include the following:

Transparent reconnection
: Clients don't have to manually reconnect to a surviving instance. You can optimally reconfigure TAF to preconnect clients to an alternate instance in addition to their primary instance when they log on. Preconnecting clients to an alternate instance removes the overhead of establishing a new connection when automatic failover takes place. For systems with a large number of connected clients, this preconnection avoids the overhead and delays caused by flooding the alternate instance with a large number of simultaneous connection requests.

Automatic resubmission of queries
: TAF can automatically resubmit queries that were active at the time the first instance failed and can resume sending results back to the client. Oracle will re-execute the query as of the time the original query started. Oracle's read consistency will therefore provide the correct answer regardless of any activity since the query began. However, when the user requests the "next" row from a query, Oracle will have to process through all rows from the start of the query until the requested row, which may result in a response lag.

Callback functions
: An application developer can register a "callback function" with TAF. Once TAF has successfully reconnected the client to the alternate instance, the registered function will be called automatically. The application developer can use the callback function to re-initialize various aspects of session state as desired.

Failover-aware applications
: Application developers can leverage TAF by writing "failover-aware" applications that resubmit transactions that were lost when the client's primary instance failed, further reducing the impact of failure. Note that unlike query resubmission, TAF itself doesn't automatically resubmit the transactions that were in-flight. Rather, it provides a framework for a seamless failover that can be leveraged by application developers.

How TAF works

TAF is implemented in the Oracle Call Interface (OCI) layer, a low-level API for establishing and managing Oracle Database connections. When the instance to which a client is connected fails, the client's server process ceases to exist. The OCI layer in the client can detect the absence of a server process on the other end of the channel and automatically establish a new connection to another instance. The alternate instance to which TAF reconnects users is specified in the Oracle Net configuration files.

Because OCI is a low-level API, writing programs with OCI requires more effort and sophistication on the part of the developer. Fortunately, Oracle uses OCI to write client tools and various drivers, so that applications using these tools can leverage TAF. Support for TAF in ODBC and JDBC drivers is especially useful; it means that TAF can be leveraged by any client application that uses these drivers to connect to Oracle. For example, TAF can provide automatic reconnection for a third-party query tool that uses ODBC. To implement TAF with ODBC, set up an ODBC data source that uses an Oracle Net service name that is configured to use TAF in the Oracle Net configuration files. ODBC uses Oracle Net and can therefore leverage the TAF feature.

TAF and various Oracle configurations

Although the TAF-Real Application Clusters combination is the most obvious combination for high availability, TAF can be used with a single Oracle instance or with multiple databases, each accessible from a single instance. Some possible configurations are as follows:

- TAF can automatically reconnect clients back to their original instances for cases in which the instance failed but the node did not. An automated monitoring system, such as the management framework under Oracle Enterprise Manager, can detect instance failure quickly and restart the instance. The fast-start recovery features in Oracle enable very short crash recovery times. Users that aren't performing heads-down data entry work can be automatically reconnected by TAF and might never be aware that their instance failed and was restarted.
- In simple clusters, TAF can reconnect users to the instance started by simple hardware failover on the surviving node of a cluster. The reconnection cannot occur until the alternate node has started Oracle and has performed crash recovery.
- When there are two distinct databases, each with a single instance, TAF can reconnect clients to an instance that provides access to a different database running in another data center. This clearly requires replication of the relevant data between the two databases.

Oracle Application Continuity

Oracle Database 12*c* introduces *Application Continuity*, an infrastructure that will replay an application request to the database when a recoverable error is received during unplanned or planned outages. During lost database sessions, the client will remain current with entered data, returned data, cached data, and variables. Where transactions are being initiated to the database, the client may not know if a commit occurred after it was issued or, if it was not issued or executed, whether it must resubmit the request upon rollback of an in-flight transaction. Restored sessions where Application Continuity is deployed include all states, cursors, variables, and the most recent transaction if there is one.

With the first release of Oracle Database 12*c*, applications can take advantage of this infrastructure using the Oracle JDBC driver as well as the JDBC Universal Connection Pool. Key components in the architecture include the JDBC replay driver, continuity director, replay context information, and Transaction Guard.

After an outage occurs, the JDBC replay driver will receive a request from the client application and send the calls in the request to the Oracle Database. The replay driver receives directions for each call from the database, and a Fast Application Notification (FAN) or recoverable error. The replay driver then obtains a new database session and checks whether the replay can progress. (Where RAC FAN provides notification of configuration and service level status, connection cleanup will include attaching to a different live RAC instance where instance failure occurred.) If the continuity directory requires a replay, then the replay driver resubmits the calls as instructed by the database. The JDBC driver will relay the response to the client application and it will simply appear to be delayed if successful. If unsuccessful, the original error is received.

Recovering from Failures and Data Corruption

Despite the prevalence of redundant or protected disk storage, media failures can and do occur. In cases in which one or more Oracle datafiles are lost due to disk failure, you must use Oracle database backups to recover the lost data.

There are times when simple human or machine error can also lead to the loss of data, just as a media failure can. For example, an administrator may accidentally delete a datafile, or an I/O subsystem may malfunction and corrupt data on the disks. Oracle's engineered systems containing Exadata Storage Servers help eliminate data corruption by being fully compliant with Oracle's Hardware Assisted Resilient Data (HARD) initiative where integrity of data blocks is checked anytime data is being written to disk.

A useful management feature for non-RAC databases used for diagnosis of on-disk data failures and repair options is the *Data Recovery Advisor*. This Advisor provides early detection of corruption through a Health Monitor, failure diagnosis, a failure impact analysis, repair recommendations, repair feasibility check, repair automation, and validation of data consistency and recoverability. It can be used with primary, logical standby, physical standby, and snapshot standby Oracle databases.

The key to being prepared to handle failures of these types are implementing a good backup-and-recovery strategy and making use of Oracle's growing number of Flashback capabilities. We'll cover those next.

Developing a Backup-and-Recovery Strategy

Proper development, documentation, and testing of your backup-and-recovery strategy is one of the most important activities in implementing an Oracle database. You must

test every phase of the backup-and-recovery process to ensure that the entire process works, because once a disaster hits, the complete recovery process *must* work flawlessly.

Some companies test the backup procedure but fail to actually test recovery using the backups taken. Only when a failure requires the use of the backups do companies discover that the backups in place were unusable for some reason. It's critical to test the entire cycle from backup through restore and recovery.

Taking Oracle Backups

Two basic types of backups are available with Oracle:

Online or hot backup
: The datafiles for one or more tablespaces are backed up while the database is active.

Closed or cold backup
: The database is shut down and all the datafiles, redo logfiles, and control and server parameter files are backed up.

With an online backup, not all of the datafiles must be backed up at once. For instance, you may want to back up a different group of datafiles each night. You must be sure to keep backups of the archived redo logs that date back to your oldest backed-up datafile, since you'll need them if you have to implement roll forward recovery from the time of that oldest datafile backup. Of course, much of this is automated today through the use of Oracle's Recovery Manager (RMAN) that is accessed via Oracle Enterprise Manager.

Some DBAs back up the datafiles that contain data subject to frequent changes more frequently (for example, daily), and back up datafiles containing more static data less often (for example, weekly). If the database isn't archiving redo logs (this is known as running in NOARCHIVELOG mode and is described in Chapter 2), you can take only complete closed backups. If the database is archiving redo logs, it can be backed up while running.

Oracle Database 12*c* introduces other possible backup strategies including multitenant container database (CDB) and pluggable database (PDB) backups. For example, you can back up a CDB and all of its PDBs or choose to back up individual PDBs. For more information about the different types of backups and variations on these types, please refer to your Oracle documentation as well as the third-party books listed.

Using Backups to Recover

Two basic types of recovery are possible with Oracle, based on whether or not you are archiving the redo logs:

Complete database recovery

If the database did not archive redo logs, only a complete closed backup is possible. Correspondingly, only a complete database recovery can be performed. You restore the database files, redo logs, and control and server parameter files from the backup. The database is essentially restored as of the time of the backup. All work done since the time of the backup is lost and a complete recovery must be performed even if only one of the datafiles is damaged. The potential for lost work, coupled with the need to restore the entire database to correct partial failure, are reasons most shops avoid this situation by running their databases in ARCHIVELOG mode. Figure 11-8 illustrates backup and recovery for an Oracle Database without archived redo logs.

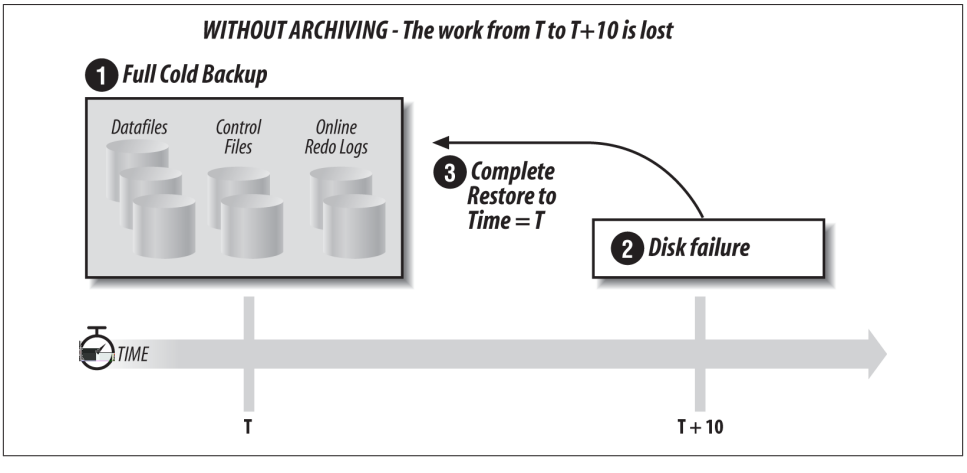


Figure 11-8. Database backup and recovery without archived redo logs

Partial or targeted restore and roll forward recovery

When you're running the Oracle database in ARCHIVELOG mode, you can restore only the damaged datafile(s) and can apply redo log information from the time the backup was taken to the point of failure. The archived and online redo logs reproduce all the changes to the restored datafiles to bring them up to the same point in time as the rest of the database. This procedure minimizes the time for the restore and recovery operations. Partial recovery like this can be done with the database down. Alternatively, the affected tablespace(s) can be placed offline and recovery can be performed with the rest of the database available. You can also restore and recover individual data blocks instead of entire datafiles. Figure 11-9 illustrates backup and recovery with archived redo logs.

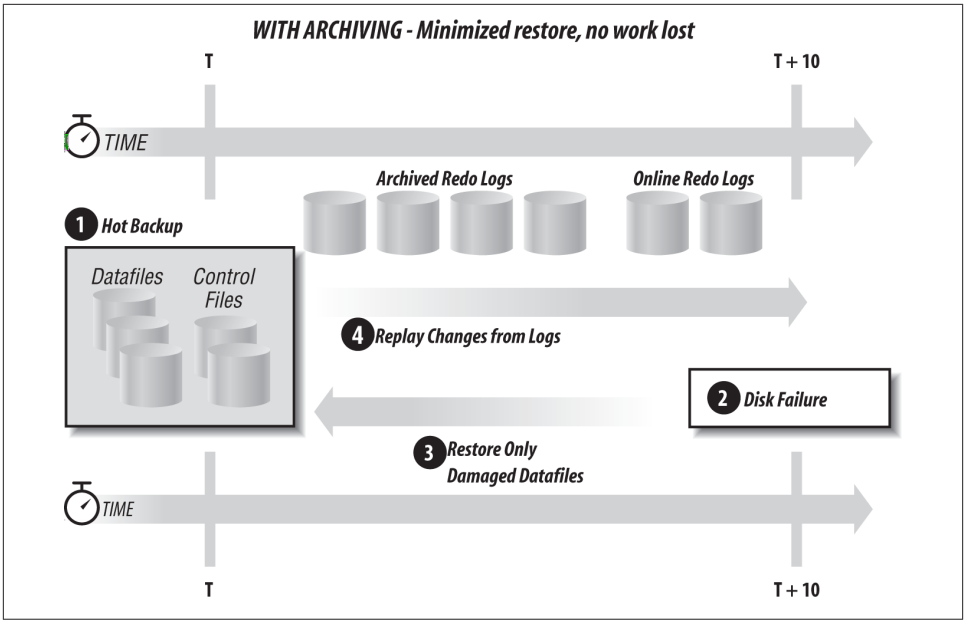


Figure 11-9. Database backup and recovery with archived redo logs

Especially useful is the LogMiner utility, accessible through Oracle Enterprise Manager, and used for investigating redo logs providing analysis for all datatypes. If the redo log has become corrupted, the LogMiner can read past corrupted records as desired in order to analyze the impact on transactions after the corruption.

Recovery Manager

Recovery Manager (RMAN) is an Oracle-recommended MAA solution for data corruption that provides server-managed online backup and recovery. RMAN does the following:

- Backs up one or more datafiles to disk or tape
- Backs up archived redo logs to disk or tape
- Backs up automatically the control file and server parameter file to disk or tape
- Backs up CDBs and PDBs to disk or tape
- Restores datafiles from disk or tape
- Restores and applies archived redo logs to perform recovery
- Restores the control file and server parameter file
- Restores CDBs and PDBs from disk or tape

- Automatically parallelizes both the reading and writing of the various Oracle files being backed up

RMAN performs the full backup operations and updates a catalog (stored in an Oracle Database) with the details of what backups were taken and where they were stored. You can query this catalog for critical information, such as datafiles that have not been backed up or datafiles whose backups have been invalidated through NOLOGGING operations performed on objects contained in those datafiles.

RMAN also uses the catalog to perform incremental backups. RMAN will back up only database blocks that have changed since the last backup. When RMAN backs up only the individual changed blocks in the database, the overall backup and recovery time can be significantly reduced for Oracle databases in which a small percentage of the data in large tables changes. Improvements in methods used by RMAN in recent Oracle releases have greatly enhanced performance for incremental backups.

RMAN reads and writes Oracle blocks, not operating system blocks. While RMAN is backing up a datafile, Oracle blocks can be written to it, but RMAN will read and write in consistent Oracle blocks, not operating system blocks within an Oracle block.

The following list summarizes the RMAN capabilities that enable high availability:

- Automated channel failover during backup and restore
- Automated failover to a previous backup during restore when the current backup is missing or corrupt
- Automated new database and temporary file creation during recovery
- Automated recovery to a previous point in time
- Block media recovery while the datafile is online
- Block change tracking for fast incremental backups
- Merged incremental backups
- Backup and restore of required files only
- Retention policy ensuring that relevant backups are available
- Resumable backup and restore if operations failed
- Automatic backup of the control file and server parameter file

Since Oracle Database 10*g*, RMAN is also used to support automated disk-based backup. Disk-based strategies have an advantage over tape: they enable random access to any data such that only changes need be backed up or recovered. RMAN can be set up to run a backup job to disk at a specific time. RMAN manages the deletion of backup files that are no longer necessary. In combination with ASM, RMAN will write all backups,

archive logs, control file autobackups, and datafile copies to a designated disk group. The single storage location is referred to as the Flash Recovery Area.

Read-Only Tablespaces and Backups

Once an Oracle tablespace is in read-only mode, it can be backed up once and doesn't have to be backed up again, since its contents cannot change unless it's placed in read/write mode. Marking a tablespace as read-only allows entire sections of a database to be marked read-only, backed up once, and excluded from regular backups thereafter.

If a datafile of a read-only tablespace is damaged, you can restore it directly from the backup without any recovery. Because no changes were made to the datafiles, no redo log information needs to be applied. For databases with significant static or historical data, this option can significantly simplify and streamline backup and restore operations.

Read-only tablespaces, combined with Oracle's ability to partition a table on a range or list of column values (for example, a date), provide powerful support for the rolling windows common to data warehouses (described in Chapter 10). Once a new month's data is loaded, indexed, and so on, the relevant tablespaces can be marked read-only and backed up once, removing the tablespaces datafile(s) from the cycle of ongoing backup and significantly reducing the time required for those backup operations.

Old-Fashioned Data Redundancy

You can also achieve data redundancy using Oracle's standard utilities. Historically, one of the most common backup methods for Oracle was simply to export the contents of the Oracle database into a file using the Oracle Export utility. This file could then be shipped in binary form to any platform Oracle supports and subsequently imported into another database with Oracle's Import utility. This approach can still provide a simple form of data redundancy if the amount of data is manageable.

Older releases featured a *direct path export* (beginning with Oracle 7.3) to avoid some of the overhead of a normal export by directly accessing the data in the Oracle datafiles. Oracle Database 10*g* and newer database releases provide a much higher speed export/import utility called *Data Pump*, and it is about 60 percent faster for export and 15 to 20 times faster for import per stream.

Another export option is to unload data from the desired tables into simple flat files by spooling the output of a SELECT statement to an operating system file. You can then ship the flat file to the secondary site and use Oracle's SQL*Loader utility to load the data into duplicate tables in the secondary database. For cases in which a significant amount of data is input to the primary system using loads, such as in a data warehouse, a viable disaster-recovery plan is simply to back up the load files to a secondary site on

which they will wait, ready for reloading to either the primary or secondary site, should a disaster occur.

While these methods may seem relatively crude, they can provide simple data redundancy for targeted sets of data. Transportable tablespaces can also be used to move entire tablespaces to a backup platform. Transportable tablespaces in Oracle Database 10*g* and newer releases let you transport tablespaces from one type of system to another without having to do an export and import of their data, increasing their flexibility for implementing redundancy, moving large amounts of data, and migrating to another database server platform.

Point-in-Time Recovery

Point-in-time datafile recovery enables a DBA to restore the datafiles for the Oracle database and apply redo information up to a specific time or System Change Number (SCN). This limited type of recovery is useful for cases in which an error occurred—for example, if a table was dropped accidentally or a large number of rows were deleted incorrectly. The DBA can restore the database to the point in time just prior to the event to undo the results of the mistake.

Point-in-time tablespace recovery allows a DBA to restore and recover a specific tablespace or set of tablespaces to a particular point in time. Only the tablespace(s) containing the desired objects need to be recovered. However, this tablespace feature needs to be used carefully, since objects in one tablespace may have dependencies, such as referential integrity constraints, on objects in other tablespaces. For example, suppose that Tablespace1 contains the EMP table and Tablespace2 contains the DEPT table, and a foreign key constraint links these two tables together for referential integrity. If you were to recover Tablespace2 to an earlier point than Tablespace1, you might find that you had rows in the EMP table that contained an invalid foreign key value, since the matching primary key entry in the DEPT table had not been rolled forward to the place where the primary key value to which the EMP table refers had been added.

Point-in-time recovery was further extended in Oracle Database 12*c*. You can now also recover tables and partitions to a specified point in time using your RMAN backups.

Flashback

Flashback is capability designed to help in recovering from errors caused by recent changes to data or tables in your Oracle database. It enables you to go back to a time to see when the data was considered valid in the database and undo damage caused in the interim. Flashback Query was the first example of this feature made available in Oracle9*i*. Today, in Oracle Database 12*c*, there are a host of Flashback capabilities. These include:

Flashback Query
: Query returns results at a specific time or System Change Number (SCN) in the past, using undo log information segments to reconstruct the data.

Flashback Version Query
: Query returns versions of rows in a table over a specified span of time.

Flashback Transaction
: Used for backing out an individual transaction and its dependent transactions by utilizing undo data to its original state.

Flashback Transaction Query
: Query returns all the changes made by a specific transaction.

Flashback Drop
: When an object is dropped it is placed in a Recycle Bin, thus enabling a user to simply un-drop the object to restore it.

Flashback Table
: A table is returned to a specific past point in time.

Flashback Restore Points
: Labels created by DBAs that map to points in time via timestamps or SCNs.

Flashback Database
: Returns the entire database to a particular point in time and can be used instead of point-in-time recovery in some situations.

Flashback Logs and Block Media Recovery
: If data blocks are corrupted, enables retrieval of more recent copies of the data blocks to speed repair.

Flashback Data Archive
: Contains transactional changes made to every record in a table for the life of the record. This capability is part of the Total Recall portion of the Advanced Compression option. To implement Total Recall, the Oracle Database creates a shadow table to hold historical changes to the main table, freeing flashback from the limitations of a fixed size to UNDO space.

Which to Choose?

All the choices we've discussed in this chapter offer you some type of protection against losing critical data—or your entire Oracle Database. But which of these is right for your needs?

To quote the standard answer to so many technical questions, "it depends." Export/import strategies provide a simple and proven method, but the time lag involved with

this method typically leaves larger time periods where data is lost in the event of a failure. Transportable tablespaces and other backups from RMAN also require time to restore. A physical standby database or RAC can offer zero data loss and faster time to recovery; however, these solutions do require the expense of redundant hardware. Asynchronous replication solutions also require redundant hardware, but while providing a great deal of flexibility, also require considerable effort in planning and coding.

You should carefully balance the cost, both in extra hardware and performance, of each of these solutions, and balance them against the potential cost of a database or server failure. Of course, any of these solutions is infinitely more valuable than not implementing any of them and simply hoping that a disaster never happens to you.

Likely, you'll deploy combinations of these solutions such as RAC for rapid instance recovery and Data Guard to enable business continuity during primary site failure. If you choose to use RAC, Data Guard, or GoldenGate, you might also use *Global Data Services*, a new configuration, maintenance, and monitoring management framework supported by Oracle Database 12*c*, for global management of high availability configurations including cloud infrastructures.

Planned Downtime

Thus far, we have focused on preventing unplanned downtime. But much of availability planning seeks to reduce planned downtime for system maintenance operations. Planned downtime is usually a concern during certain system changes, data changes, and application changes. Today, much of this downtime has largely disappeared because of Oracle's extensive online management capabilities.

For example, changes that can be made while a system is operational include hardware upgrades, operating system upgrades, patching, and storage migration. Where RAC configurations leverage ASM, rolling upgrades (first introduced in Oracle Database 10*g* Release 2) can take place with no downtime. In fact, Oracle's engineered systems that run Oracle Databases are designed to take advantage of this.

Oracle Data Guard might be used to minimize downtime for system, database, and patch set upgrades where Oracle databases are not in RAC configurations. It offers a logical standby database capability where the standard Oracle archive logs are transformed into SQL transactions, and these are applied to an open standby database. The logical standby database can also be different physically from the primary database and can be used for different tasks. For example, the primary Oracle database might be indexed for transaction processing while the standby database might be indexed for data warehousing. Although physically different from the primary database, the secondary database is logically the same and can take over processing in case the primary fails. As archive logs are shipped from the primary to the secondary, undo records in the shipped archive log

can be compared to the logical standby undo records to guard against potential corruption.

Pluggable databases in Oracle Database 12*c* offer a dramatic potential solution for reducing downtime for upgrades. The stated direction of Oracle is to allow users to upgrade the multitenant container database (CDB) only, and simply unplug a pluggable database from one container and move it to an upgraded multitenant container database, which could significantly reduce this type of planned downtime.

For data changes, Oracle provides online reorganization capabilities in recent releases, a task that often required extensive planning in the past. Years ago, discussions of how to reorganize the Oracle database were popular topics at most Oracle database conferences. Today, you can easily modify physical attributes of a table and change data and table structure while users are accessing the data. Also supported are online index builds and online moves of partitioned tables.

Edition-based redefinition, discussed in Chapter 4, enables an Oracle database upgrade to occur while an application is in use that accesses the database. In some cases, when the new edition is available, a rollover to that edition can occur with no downtime. Other online application maintenance and upgrade capabilities for the database, when deployed under applications, include use of GoldenGate for rolling upgrades and ability to set invisible indexes and invisible columns.

Many of the Oracle features and options we've covered in this chapter are also useful for speeding database migrations and minimizing possible downtime during the migration process. For example, migrations to new servers can be greatly aided by Data Guard, Data Pump, RMAN support of transportable tablespaces, and GoldenGate. Migrations to new storage are also aided by using transportable tablespaces with RMAN and GoldenGate.

Achieving RAC Benefits on Single Nodes

You might want some of the benefits of RAC such as online migration of Oracle database instances, patching, and upgrading of the operating system and the Oracle database with no downtime, but wish to deploy your Oracle database to a single node in a cluster. This is precisely the capability that Oracle RAC One Node provides. As with RAC, RAC One Node fully supports Data Guard. It can also be used to provide simple failover to a second node and provides a means to rapidly upgrade to multinode RAC configurations.

CHAPTER 12
Oracle and Hardware Architecture

In Chapter 2, we discussed the architecture of the Oracle Database, and in Chapter 7, we described how Oracle uses hardware resources. Your choice of servers and storage will ultimately help determine the scalability, performance tuning, management, and reliability options you will have. Where systems are configured without consideration of the proper balance of CPUs, memory, and I/O for projected workloads, database management and tuning becomes more complicated and limited in effectiveness.

Over the years, Oracle developed new features in its flagship Enterprise Edition Database to address evolving platform strategies, including clustered systems, grid computing, and cloud deployment. In 2008, Oracle introduced its first engineered system, the Oracle Exadata Database Machine, featuring Database Server nodes, Exadata Storage Server cells, and Oracle Database 11*g* with Exadata Storage Server software that are uniquely optimized to work together. This chapter provides a basis for understanding how Oracle Database 12*c* can take advantage of commodity systems and each of the Oracle engineered systems on which the Oracle Database is supported. The engineered systems most often used for deploying the Oracle Database are the following:

- Oracle Exadata Database Machine
- Oracle SuperCluster
- Oracle Database Appliance

First, we'll discuss systems basics and the impact on performance. This portion of the chapter will be relevant for deploying commodity servers and storage, as well as providing a basis for understanding some of the optimizations provided by Oracle's engineered systems. We'll then provide an introduction to the varieties of Oracle engineered systems. When you complete this chapter, you should have a good understanding of the server and storage choices you have.

System Basics

We will begin with a review of the components that make up any hardware platform and the impact these components have on the overall platform. You'll find the same essential components under the covers of any computer system:

- Nodes or systems consisting of CPUs (with multiple cores) for executing the basic instructions that make up computer programs and memory, storing recently accessed instructions and data
- Storage or storage cells that typically consist of a combination of disk storage and device controllers, sometimes also including PCI Flash (such as is present in Exadata Storage Server cells)
- Interconnects for distributing data to nodes and storage
- Network ports enabling connectivity for business usage and administration networks

The number, power, throughput, and capabilities of the individual components determine the ultimate cost and scalability of a system. Systems and nodes with more processors and cores are typically more expensive and capable of doing more work than systems and nodes with fewer; systems and nodes containing newer versions of components typically deliver superior capabilities at similar or lower price points than older versions.

Each system component has its own performance characteristics, including a time to access and transport data, or a *latency cost*. The latency cost of a component is the amount of latency the use of that component introduces into the system; in other words, how much slower each successive level of a component is than its previous level (e.g., Level 2 versus Level 1; see Table 12-1). Each component also has limited capacity and most components can use other resources when the demand for their capacity is exceeded.

The CPU and the Level 1 (L1) memory cache on the CPU have the lowest latency, as shown in Table 12-1, but also the least capacity. Disk has the most capacity but the highest latency. Note that the capacities for the storage components marked by the * are typical in balanced server and storage configurations optimized for performance.

There are several different types of memory: an L1 cache, which is on the CPU chip; an L2 (Level 2) cache on the CPU surface, an L3 cache on the same board as the CPU (not as widely offered); and main memory, which is the remaining memory on the system accessible through the memory bus.

Table 12-1. Typical sizes and latencies of system components

Element	Typical storage capacity	Typical latency
CPU	None	None
L1 cache (on each CPU)	128 KBs	1 nanosecond
L2 cache (on each CPU surface)	1.5 MBs	4 nanoseconds
L3 cache (on each CPU board)	12 MBs	10 nanoseconds
Main memory (per node)	100+ GBs	75 nanoseconds
Smart Flash Cache*	0.5 TBs	50,000 nanoseconds
Disk (per node)*	10–50 TBs	4-7 million nanoseconds

An important part of tuning any Oracle database involves reducing the need to read data from sources with the greatest latency (e.g., disk) and, when a disk must be accessed, ensuring that there are as few bottlenecks as possible in the I/O subsystem. As the Oracle database accesses a growing percentage of its data from memory rather than disk, the overall latency of the system is correspondingly decreased and performance is improved. The Smart Flash Cache in Exadata Storage Server cells is important in that it can transparently provide a cache for frequently accessed data in the storage tier of the platforms it is available for.

Today's multicore processors, integrated circuits that contain multiple processors, enable simultaneous processing of multiple tasks. Processor developers are racing to provide more cores in CPUs to differentiate themselves.

Each thread in a server operating system can be used to support a concurrent process, which can execute in parallel. Oracle can determine the degree of parallelism when running DML, DDL, and queries based on the execution cost using the automatic degree of parallelism feature, described in Chapter 7. This adaptive multiuser feature makes use of algorithms that take into account the number of threads. Parallel statement queuing enabled using the Database Resource Manager can give the right priorities and number of parallel processes to consumer groups in complex multiuser environments. Additional tuning parameters can also affect parallelism, although the need for tuning of such parameters is much diminished in recent Oracle releases.

Symmetric Multiprocessing Systems and Nodes

Early systems, described as uniprocessor, contained single CPUs with their power limited by the ultimate speed of that processor—and all applications had to share this one resource. Symmetric Multiprocessing (SMP) systems were invented to overcome this limitation by adding CPUs to the memory bus and are common in today's systems and nodes. Figure 12-1 represents such a node.

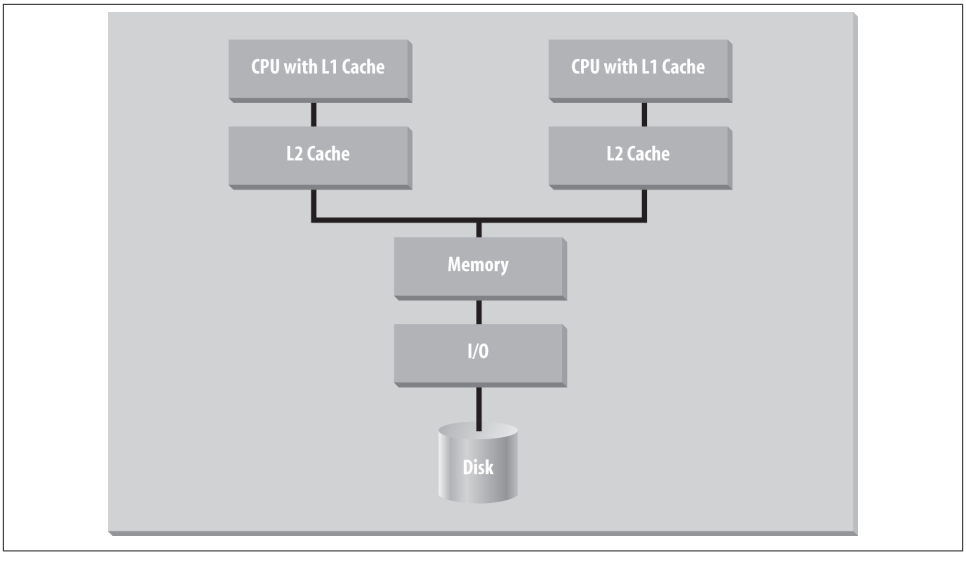


Figure 12-1. Typical Symmetric Multiprocessing (SMP) node

Each CPU has its own memory cache. Data resident in the cache of one CPU is sometimes needed for processing by a second CPU. Because of this potential sharing of data, the CPUs for such machines must be able to "snoop" the memory bus to determine where copies of data reside and whether the data is being updated. This snooping is managed transparently by the operating system that controls the SMP system. Oracle Standard Edition One, Standard Edition, or Enterprise Edition can be used on these platforms. SMP platforms have been available since the 1980s as midrange systems, primarily as Unix-based machines. Today, the most popular operating systems in this category are Linux and Windows variations, though Solaris, AIX, and HP-UX also remain popular. High-end systems and nodes feature CPUs containing high numbers of cores, larger L2 cache, a faster memory bus, and/or multiple higher-speed I/O channels. Each enhancement is intended to remove potential bottlenecks that can limit performance.

The number of CPUs possible in an SMP system or node is limited by scalability of the system (memory) bus. As more CPUs are added to the bus, the bus itself can become saturated with traffic between CPUs attached to the bus. Memory itself can now scale into the terabytes (TBs) on such systems.

Of course, the database must have parallelization features to take full advantage of the SMP architecture. Oracle operations such as query execution and other DML activity and data loading can run as parallel processes within the Oracle server, allowing Oracle to take advantage of the benefits of multiprocessor systems. Oracle, like all software systems, benefits from parallel operations, as shown by "Amdahl's Law."

Total execution time = (parallel part / number of processors) + serial part

Amdahl's Law, formulated by mainframe pioneer Gene Amdahl in 1967 to describe performance in mixed parallel and serial workloads, clearly shows that moving an operation from the serial portion of execution to a parallel portion provides the performance increases expected with the use of multiple processors. In the same way, the more serial operations that make up an application, the longer the execution time will be because the sum of the execution time of all serial operations can offset any performance gains realized from the use of multiple processors. In other words, you cannot speed up a serial operation or a sequence of serial operations by adding more processors.

Each subsequent release of Oracle has added more parallelized features to speed up the execution of queries as well as the tuning and maintenance of the database. For an extensive list of Oracle operations that can be parallelized, see the section "What Can Be Parallelized?" on page 187 in Chapter 7.

Oracle's parallel operations take advantage of available CPU resources. If you're working with a system on which the CPU resources are already being completely consumed, this parallelism will not help improve performance; in fact, it could even hurt performance by adding the increased demands for CPU power required to manage the parallel processes. Oracle's automatic degree of parallelism and management of consumer groups using the Database Resource Manager can help prevent this situation.

Clustered Solutions, Grid Computing, and the Cloud

Clustered systems have provided a highly available and highly scalable solution since initially appearing in the 1980s in a DEC VAXcluster configuration. Clusters can combine all the components of separate machines, including CPUs, memory, and I/O subsystems, into a single hardware entity. However, clusters are typically built by using shared disks linked to multiple "nodes" (computer systems). A high-speed interconnect between systems provides a means of exchanging data and instructions without writing to disk (see Figure 12-2). Each system or node runs its own copy of an operating system and Oracle instance. Grids, described later in this chapter, are typically made up of a few very large clusters.

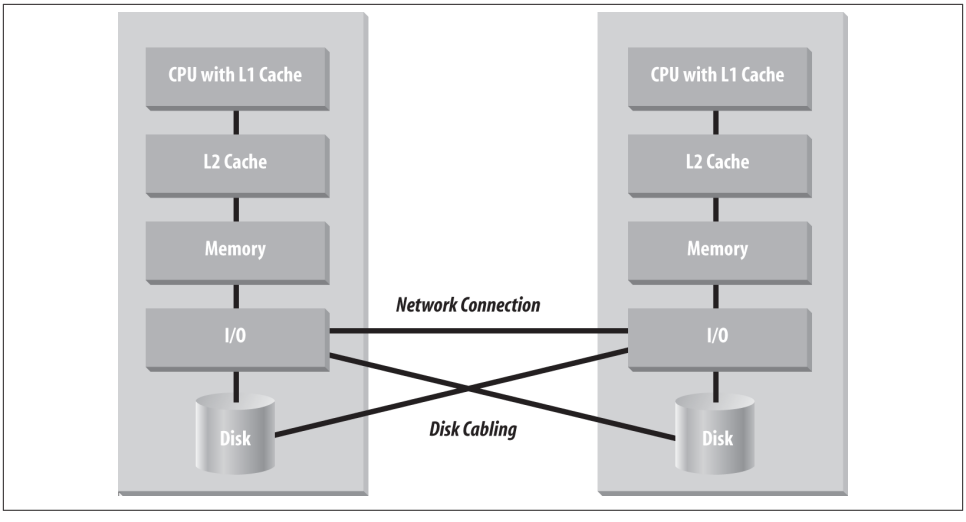


Figure 12-2. Typical cluster (two nodes shown)

Oracle's support for clusters dates back to the VAXcluster. Oracle provided a sophisticated locking model so that the multiple nodes could access the shared data on the disks. Clusters require such a locking model because each machine in the cluster must be aware of the data locks held by other, physically separate machines in the cluster.

That Oracle solution evolved into Real Application Clusters (RAC), replacing the Oracle Parallel Server (OPS) that was available prior to Oracle9*i*. RAC is used when deploying the Oracle Database on generic Windows, Linux, or Unix-based clusters and on Oracle's engineered systems. Oracle provides an integrated lock manager that mediates between different servers, or nodes, that seek to update data in the same block.

RAC introduced full support of Cache Fusion, where data blocks are shipped from one node's cache to another, eliminating the latency of disk-based cache coordination. Cache Fusion is different from the standard locking mechanisms that are described in Chapter 8, in that it applies to blocks of data, rather than rows. The mediation is necessary since two different nodes might try to write to different rows in the same physical block, which is the smallest amount of data that can be used by Oracle.

Cache Fusion initially greatly improved performance for read/write operations compared to the previous OPS and later added improved performance for write/write operations in Oracle9*i* RAC. Today, Oracle supports Sockets Direct Protocol (SDP) and asynchronous I/O protocols, lighter-weight transports than those used in previous traditional TCP/IP-based RAC implementations, which resulted in lower resource usage and better performance for RAC systems. More recent database releases further improved performance, supporting faster interconnects such as InfiniBand using Reliable Datagram Sockets (RDS).

The "*g*" in Oracle nomenclature for Oracle Database 10*g* and Oracle Database 11*g* signified a focus on enabling grid computing. *Grids* are simply pools of computers that provide needed resources for applications on an as-needed basis. The goal is to provide computing resources that transparently scale to the user community, much as an electrical utility company can deliver power to meet peak demand by accessing energy from other power providers' plants via a power grid. Computing grids enable this dynamic provisioning of CPU and data resources.

The Oracle Database with RAC forms the foundation for the provisioning of these resources. Today, such configurations are increasingly deployed in on-premise private clouds or to provide off-premise resources in public clouds. The "*c*" in Oracle nomenclature for Oracle Database 12*c* indicates Oracle's growing focus here.

The Oracle Database with RAC forms the foundation used in provisioning of resources, enabling dynamic load balancing and workload management. Key features in Oracle Databases and related products further supporting such deployment models include:

Pluggable Databases
: Introduced with Oracle Database 12*c*, this feature enables many pluggable databases (PDBs) to exist within single Oracle database occurrences (called multitenant container databases, or CDBs) and is useful when consolidating larger numbers of databases to a single platform than otherwise might be supported on a given platform configuration, provisioning of new databases, copying existing databases, and for fast patching and upgrades. Please refer to Chapter 2 for a more detailed discussion of pluggable databases.

Active Data Guard
: Active Data Guard is typically used for high availability implementations and enables the creation of farms of read-only databases. The Data Guard Broker enables creation, management, and monitoring of these configurations. Please refer to Chapter 11 for a more detailed discussion of Active Data Guard.

GoldenGate
: GoldenGate enables replication of updates in near real-time among multiple databases. GoldenGate is also covered in more detail in Chapter 11.

Enterprise Manager 12c
: Enterprise Manager 12*c* manages cloud and grid infrastructures from a central location, including Oracle RAC Databases, storage, Oracle Application Servers/Fusion Middleware, network services, and Oracle-engineered systems and storage. See Chapter 5 for a discussion regarding managing your Oracle database and engineered system using Enterprise Manager.

Disk and Storage Technology

The discussion of hardware architectures in this chapter has thus far centered on system resources such as CPUs, memory, and I/O subsystems, and noted that parallelism can take advantage of these resources. An important way to increase hardware performance is to tune for optimal I/O performance, which includes spreading data across disks and providing an adequate number of access paths to the data. Since access to disk has the greatest latency, another focus of I/O tuning is keeping as much data as possible retrieved from disk in-memory.

On systems designed for performance, disks are often directly attached to nodes or via a high-speed interconnect such as InfiniBand (in many of Oracle's engineered systems). Network Attached Storage (NAS) and Storage Area Networks (SAN) provide cost-effective alternatives but with performance trade-offs. Disks are configured in a variety of ways for redundancy, eliminating the possibility of single points of disk failure resulting in loss of access to data.

Disk is commonly deployed in arrays, the industry standard being RAID (Redundant Array of Inexpensive/Independent Disks). You can use RAID as a part of any of the configurations we've discussed to provide higher performance and reliability. RAID disk arrays were introduced in this book in Chapter 7 and discussed in the context of their use in high availability scenarios in Chapter 11. Please refer to those chapters for more information about RAID. In addition, since Oracle Database 10*g*, Automatic Storage Management (ASM) delivers much of the functionality of a RAID array, such as striping and mirroring, with a collection of commodity disks. ASM is further described in Chapter 5. The storage in an Oracle-engineered system running the Oracle database is typically mirrored once (e.g., two copies) for normal redundancy and twice (three copies) for a higher degree of redundancy.

Oracle9*i* first introduced table compression in the Oracle database as a means of decreasing disk storage requirements, primarily in data warehousing. Duplicate values in a data block are eliminated because values that are duplicated are stored in a symbol table at the beginning of the block, and all additional occurrences are replaced with a short reference to the symbol table. Oracle Database 11*g* introduced an Advanced Compression Option for insert, update, and delete operations important in OLTP operations. Data compression of three to four times is commonly observed today. Exadata Storage Servers added support for Hybrid Columnar Compression, described below. In addition to reducing disk storage, compressed data can also be advantageous for performance when it allows a set of data to fit entirely into cache (instead of requiring disk access).

Since disk capacities are constantly growing with newer disks available at lower cost, many organizations are now storing all relevant data online in disk storage for data warehousing and business intelligence implementations. Given that disks delivering the best performance are typically more expensive and of lower capacity, many now deploy

such disks in combination with higher capacity but lower performing (and cheaper) disks for less frequently accessed data. Information Lifecycle Management (ILM) in the Oracle database, particularly the ILM Assistant (first made available in 2007), provides the capability to manage such an environment.

Oracle's Engineered Systems

Oracle refers to proper configurations—those that feature proper I/O (especially spindles that provide access paths to storage), memory, and CPUs—as *balanced configurations*. IT organizations found working with multiple hardware vendors in order to get proper storage configurations difficult. This led Oracle to develop reference configurations with several key hardware platform and storage vendors to help provide more accurate initial sizing.

However, many IT organizations still found integration of the components to be time-consuming, and such configurations often became unbalanced over time as server and storage modifications were made. As complexity grew, organizations also found the time to debug problems increasing partly due to multiple platform vendors being involved. Oracle came to realize the need to create and support engineered systems with predefined configurations and upgrade paths. The first such system introduced was the Oracle Exadata Database Machine. Today, Oracle offers a family of engineered systems that can be deployed as individual solutions or in combination. Our focus in this chapter is on engineered systems designed for the Oracle database.

All of the systems share some common traits. All are built from Oracle Sun servers that feature redundant power supplies and multiple interconnect and network connections. They feature GbE and 10 GbE ports providing connectivity to user networks. All of the Sun servers also feature Integrated Lights-Out Management ports that can send Automatic Service Requests to Oracle Support when components fail or are about to fail. The systems rely on RAC providing high availability for nodes and mirroring for high availability storage. All are managed using Oracle Enterprise Manager, including the hardware management extensions provided by Sun Operations Center components.

We'll next explore some of the unique characteristics of each system.

Oracle Exadata Database Machine

As this edition of *Oracle Essentials* was published, the Oracle Exadata Database Machine was far and away the most popular Oracle-engineered system to deploy Oracle databases on. Initially introduced for data warehousing, Exadata is also widely used in hosting online transaction processing (OLTP) databases, and mixed workload databases, and as a consolidation platform where multiple OLTP and data warehousing databases are deployed. The systems are designed to run the Oracle Database 11*g* Enterprise Edition or more current releases. (The releases that are supported are dependent on Oracle

database releases available when specific models were introduced. Upgrades to newer Oracle database releases are supported on older platforms.) The Linux operating system has proven to be the most popular choice on these platforms, though some models also support Solaris.

Details of the server and storage configurations change as Oracle releases new versions of Exadata. All Oracle Exadata Database Machine versions consist of Database Server nodes and Exadata Storage Server cells and house at least two InfiniBand switches. A third switch, called a spine switch, is provided on configurations designed to scale beyond a single Rack. The Database Server nodes are configured with two CPUs (especially appropriate for Oracle database applications that scale best with RAC) or eight CPUs (especially appropriate for database applications that scale best on SMP systems). Each node contains memory, local storage for the Oracle Database Server node software, InfiniBand, and Ethernet connections. Ethernet connections are used for administration (Exadata contains its own Ethernet switch for the administration network) and for user access to the databases. Eight dual-processor nodes or two 8-processor nodes make up a Full Rack configuration.

Exadata Storage Server cells each contain 12-high performance or high-capacity disk drives, Flash storage in the form of PCI cards, and InfiniBand connections. A Full Rack contains 14 Storage Server cells. With today's disk capacities, a Full Rack can contain in excess of 500 TB of disk. The Smart Flash Cache storage provides an intermediate area (measuring into the TBs) that is used by the database for a cache between memory and storage. Objects can be pinned into the Flash Cache using the ALTER TABLE statement. You can also create Flash Disks using this Flash Cache, keeping in mind that this is a volatile area (so a backup strategy should also be put in place). Figure 12-3 illustrates an Oracle Database Machine Full Rack configuration.

Where nodes containing two CPUs are deployed, partial Rack configurations are available in similar ratios of Database Server nodes to Exadata Storage Server cells as present in the Full Rack. For example, a Half Rack contains four Database Server nodes and seven Exadata Storage Server cells. For scaling beyond a single Full Rack, the InfiniBand spine switch in the Rack provides enough free ports for connecting up to eight Full Racks together without the need for another external switch.

The Exadata Storage Server software provides Oracle database optimization not available for non-Exadata storage devices. This software can implement smart scans, which occur in storage and utilize the CPUs and memory in the Exadata Storage Server cells for selection and other operations. Other Oracle database functionality is also pushed to the Exadata Storage cells and performed in parallel. Hybrid Columnar Compression (HCC) transparently organizes and stores data by table column increasing compression ratios, typically by 10 times for data warehouses, thus improving scans. Individual row organization is self-contained in compression units so minimal I/O is required to re-

trieve an entire row. In archival mode, compression ratios of 15 times are typical. During direct load operations, data is transformed into columnar format and compressed.

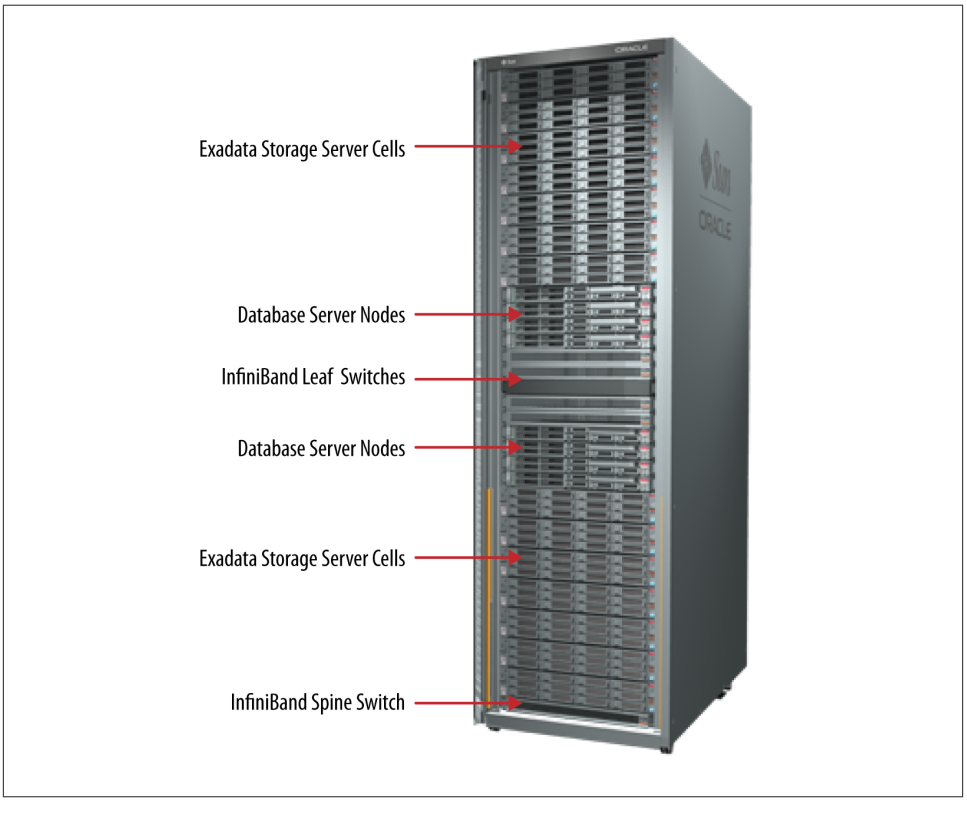


Figure 12-3. Oracle Exadata Database Machine

Other Exadata Storage Server software capabilities include smart scan offload for the aforementioned HCC, smart scan offload for tables with more than 255 columns, smart scan offload for encrypted tablespaces and columns, and offload to storage of data mining scoring and other statistics.

In addition to performance benefits obtained from the Exadata Server Software, faster querying of fully encrypted databases is possible because decryption processing is moved into the processors in the Exadata Storage Server cells. SQL monitoring support and I/O performance graphs for Exadata Storage are provided by Enterprise Manager Performance Diagnostics.

Oracle Exalogic

First introduced as Oracle's Elastic Cloud platform, Oracle Exalogic is optimized to run applications, especially those that are built upon middle-tier Java servers. The Exalogic Elastic Cloud Software includes an Exalogic Base Image and System Utilities (device drivers, firmware, software libraries, and configuration files for Exalogic), an Oracle Traffic Director for load balancing, and an Oracle Virtual Assembly Builder with restricted use licenses of the WebLogic Server and Coherence. The Exalogic hardware features compute nodes and storage, an InfiniBand interconnect, and an Ethernet switch for administration. It has less disk capacity and Flash than Exadata and doesn't contain Exadata Storage. Figure 12-4 illustrates an Oracle Exalogic configuration. Exalogic can be expanded to multiple racks.

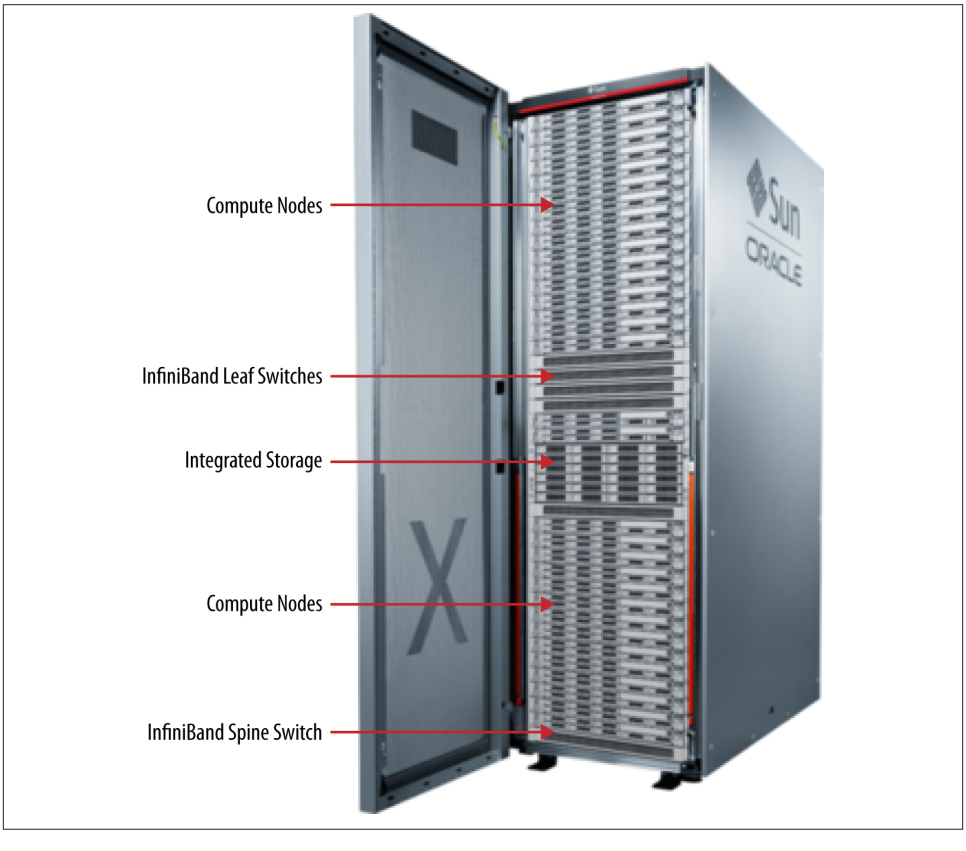


Figure 12-4. Oracle Exalogic

Exalogic is often deployed to consolidate middle-tier WebLogic Servers to a single platform and as a frontend to Oracle Exadata Database Machines via InfiniBand or other

servers via Ethernet. However, as Exalogic is deployed with either Linux or Solaris operating systems, it is sometimes used to also host Oracle databases, and some organizations use it to host both the middle tier and backend components of an application.

Oracle SuperCluster

The SuperCluster is designed to provide a general purpose engineered system to organizations that currently deploy or are planning to deploy Sun SPARC T-class platforms and the Solaris operating system. Similar to Exalogic, it can be used to host middle-tier applications and can run the Exalogic Elastic Cloud Software. Like the Oracle Exadata Database Machine, it features Exadata Storage Server cells and supports Oracle database versions that work with the Exadata Storage Server software. The SuperCluster also features fiber ports enabling the connection of SAN-based non-Exadata storage and can therefore run a wide variety of Oracle database versions that existed before Exadata. In addition, the system contains a ZFS Storage Appliance for near-line storage. Figure 12-5 illustrates a typical SuperCluster configuration.

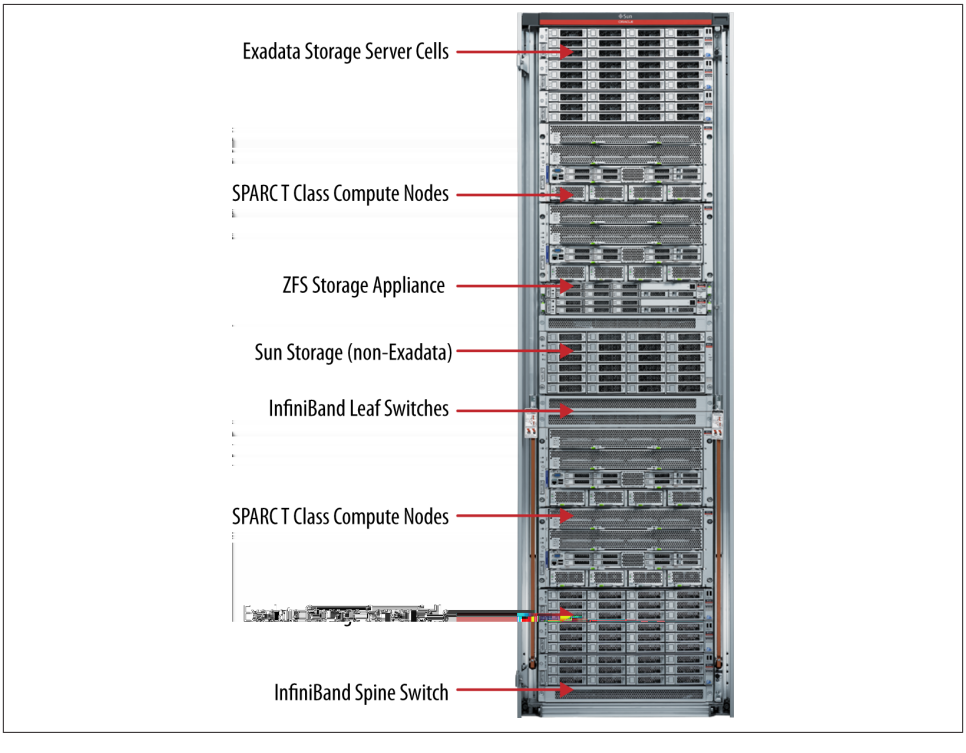


Figure 12-5. Oracle SuperCluster

Similar to Exadata and Exalogic, the interconnect switch is InfiniBand and there is an Ethernet switch and network for administration. It can be expanded with additional SuperCluster Racks and Exadata Storage.

Thus, the SuperCluster supports the most varied types of use cases. It can provide a complete system in a smaller footprint than you can achieve with separate components. The trade-off is that the SuperCluster offers less flexibility in configuration options than choosing individual engineered systems that each have been targeted and optimized for unique workloads (e.g., Oracle database or middle tier).

Oracle Database Appliance

The Oracle Database Appliance (see Figure 12-6) is a much smaller configuration than the Oracle Exadata Database Machine or SuperCluster. It is designed to meet the database needs of mid-sized and smaller businesses or to serve as a departmental server. Consisting of two server nodes in a rack-mountable 4U chassis, the nodes are linked via redundant GbE connections. The system features a simple three-step installation process consisting of plugging in the power supplies, connecting the nodes into the network, and following a wizard-driven install when the nodes are first powered up.

The nodes each contain 10 disk drives for database data storage and 4 disk drives for redo logs. The nodes run the Linux operating system and each can run Oracle's RAC One software, or more commonly are configured with Oracle RAC for scalability and availability. Unlike other engineered systems supporting the Oracle database, the Database Appliance cannot be scaled beyond the two server nodes. Exadata Storage Server software is not supported on the Oracle Database Appliance.

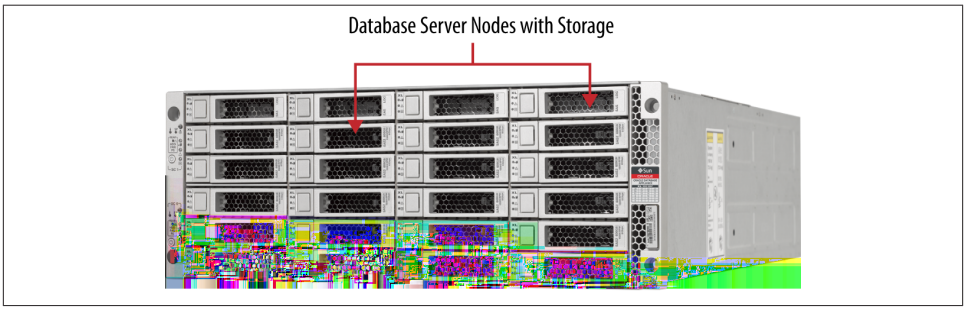


Figure 12-6. Oracle Database Appliance

Other Engineered Systems

Oracle's engineered systems extend beyond the four systems described above. Other systems are designed to support very different workloads from those typical in Oracle Databases. The Oracle Exalytics In-Memory Machine provides a large middle-tier

memory footprint and large number of cores to drive advanced visualization and performance for Oracle's Business Intelligence Foundation Suite. Exalytics can also be deployed as the platform under Oracle's data discovery tool, Endeca, or to run the Hyperion Planning applications.

The Oracle Big Data Appliance is configured to deliver an optimal platform for Cloudera's Hadoop distribution. Oracle's NoSQL Database can also be deployed on this platform. Both Exalytics and the Big Data Appliance have InfiniBand interconnects that can provide linkage to Exadata, the SuperCluster, and Exalogic, building out an analytics footprint. We describe Oracle's data warehousing and analytics footprint in much more detail in Chapter 10.

Choosing and Defining the Right Platform

As previously mentioned, choosing and defining the right custom platform can be difficult. Coordination among individuals skilled in server hardware, storage, networks, and Oracle database software is often a piecemeal undertaking requiring multiple iterations, sometimes only to be negated by budget limitations and policies. Where multiple vendors' products are involved, arrival of ordered components is just the start of integration and testing. In most organizations, this process takes six to nine months, often with mixed results. When something breaks, problem resolution requiring the coordination of multiple vendors can also be time-consuming. We'll cover justifying an Oracle-engineered system later in this section partly based on these considerations, but first we'll touch on sizing and availability planning.

Sizing and Planning for Growth

Initial sizing and configuring of hardware platforms in most organizations is based on considering a combination of the following: data storage volume needs, performance required in order to solve business problems, matching IT service level agreements (SLAs) for both availability and performance, required memory where multiple Oracle databases are to be deployed, and costs of alternatives. Sizing a platform for a single transactional application or data warehousing workload can be readily accomplished by starting with a profile of the current workload characteristics and projecting future changes. Oracle has sizing tools for each of the engineered system platforms mentioned in this chapter and can assist in this exercise.

Sizing for platforms where multiple Oracle Databases are deployed can be trickier. For example, should the platform be sized around peak workloads in each Oracle database? Do the peak workloads occur at the same time in each database? Are availability requirements similar across the databases or very different? Depending on how you answer these questions, consolidation of multiple databases to a single system may be relatively easy or quite complicated.

As we noted in Chapter 5, Enterprise Manager provides access to a number of database utilities for managing concurrent workloads, including on platforms running multiple databases. Database server and connection pool configurations can help control resource utilization. The Database Resource Manager enables restriction of resources available to users by placing them in consumer groups with defined limitations. For example, placing upper limits on the Database CPU_COUNT will limit the number of CPU cores available for a single database instance, known as "instance caging." The Database I/O Resource Manager is available for Exadata platforms and can segregate low-priority workloads from those of higher priority to keep them from flooding the disk queue when higher priority workloads are being executed. Quality of Service (QoS) management enables policy-driven resource allocation using data gathered from the entire Oracle stack, preventing such problems as memory starvation.

As you size your system, you are no doubt aware of the following truism—the longer you wait to buy your platform, the cheaper it will get for similar or even better levels of performance. According to Moore's Law, credited by Intel to Gordon Moore in 1965 (and proven many times over since then), each chip will double in computing power every 18–24 months, each time providing huge leaps in performance. Today, such performance increases are driven by increased clock speeds and the introduction of more cores in the processors, and have been demonstrated in configuration changes that have occurred on platforms such as the Oracle Exadata Database Machine.

This continual increase in performance characteristics often at little or no change in price is an ongoing fact of life in the computer hardware industry. You should buy what you need, when you need it, and plan for the obsolescence of hardware by recycling it into the organization for less demanding workloads when it no longer meets the needs of an individual application. For instance, today's departmental server may turn into tomorrow's web server. If you've deployed Oracle's larger engineered systems, you have more flexibility since you can continue using older nodes and Storage Server cells as part of the existing computing footprint, either by filling out partially full Racks or connecting older Racks to new via InfiniBand.

Maximum Availability Architecture Considerations

While we have noted that there is a high degree of redundancy in engineered systems, configuring for planned and unplanned downtime is still a best practice. You should maintain separate development, test, and production systems in an enterprise class infrastructure. Where engineered systems are deployed, Oracle is able to monitor support requests across a large number of commonly configured systems in varied customers, and issue patches proactively in patch sets. The patch sets often fix problems you have not yet encountered, but it is a good idea to remain current as this will help you deliver much better levels of service. Typically, predefined suites of test cases are run on newly patched test systems before the patch sets are deployed into production. Oracle's Real Application Testing tool can be useful here.

Chapter 11 covers high availability considerations in much more detail. Depending on business case, you might choose to simply deploy Real Application Clusters for high availability or develop a complete disaster recovery plan that includes Data Guard. Disaster recovery distances for Oracle-engineered systems connected by InfiniBand were limited to distances of 100 meters prior to the introduction of repeaters.

Part of such a strategy often includes additional hardware components including storage expansion racks and backup devices. For example, Exadata Storage Expansion Racks, holding up to 18 Exadata Storage Server cells, can be configured to support an online ILM deployment scenario when connected via InfiniBand to the Oracle Exadata Database Machine or SuperCluster. The Sun ZFS Storage Appliance provides "near-line" storage for RMAN-initiated backup and recovery and can provide over a petabyte of storage. When connected via InfiniBand, it has demonstrated NFS-based backup performance of over 20 TB per hour and restoration rates of over 9 TB per hour. Where larger capacities must be backed up, Oracle has tape libraries that can offer practical solutions.

Justifying an Oracle Engineered System

Engineered systems are generally implemented during deployment of new projects, to overcome issues caused by the current infrastructure, or for purposes of data center consolidation. Applications deployed on other platforms running Oracle can move unchanged onto Oracle's engineered systems since the Oracle database versions are consistent and platform optimizations are transparent across the various Oracle-supported hardware platforms. So, Oracle mandates no special certification process for applications to run on the Oracle-engineered systems other than running on supported database versions for those platforms. A few applications providers have mandated their own certification process, however, and it is a good idea to check with your application providers regarding their policies and Oracle database versions supported.

Commonly cited reasons for choosing to deploy engineered systems instead of traditional commodity servers, storage, and components include:

- Reduced time to system delivery
- Improved Oracle database performance enabling new business revenue growth or cost savings
- Reduced total cost of power
- Reduced floor space footprint
- Reduced cost and complexity of management
- Improved time to problem resolution

Time to system delivery is essentially the time from system order to delivery and installation, typically about four to six weeks. This becomes especially important if delivery of the system will enable new business usage that drives revenue growth or cost savings, since the sooner these occur, the sooner return on investment can be reached. The packaging of multiple nodes and storage in single frames often reduces overall power consumption and floor space footprint in the data center. System management cost (and time to make changes) is reduced when database administrators take on the role of managing these systems via Enterprise Manager, including managing the Oracle Databases, operating system, and storage. As Oracle supports the entire footprint, time to problem resolution is also reduced, enabling an organization to better meet or exceed service level agreements.

An interesting aspect of the engineered systems that contain CPUs in both the server nodes and Exadata Storage Server cells is how Oracle prices Oracle Database licenses. Only CPU cores on the server nodes count in the pricing equation. Of course, Exadata Storage Server software is priced separately, but many organizations find the trade-offs in license pricing and performance to be in their favor when comparing this model to their traditional commodity servers and storage.

A cost-benefit analysis is frequently part of the process of justifying the purchase and deployment of a new engineered system. Some or all of the potential benefits we just mentioned can lead to justification. However, your analysis should also include cost associated with change management and training that is common when moving to a new platform. It is likely that this process will go a long way toward determining whether an engineered system makes sense for your initiative.

CHAPTER 13

Oracle Distributed Databases and Distributed Data

Data in large and mid-sized companies can sometimes be spread over many different databases. The data can be on different servers running different operating systems or even different database management systems. The data needed to answer any specific business question may need to be accessed from more than one server. A user may need to access this separate data on several servers simultaneously, or the data required for an answer may need to be moved to a local server. Inserts, updates, or deletions of data across these distributed servers may also be necessary.

There are two basic ways to deal with data in distributed databases: as part of a single distributed entity in which the distributed architecture is transparent, or by using a variety of replication or data transportation techniques to create copies of the data in more than one location. This chapter examines these options and the technologies associated with each solution.

Of course, there can be performance challenges when there is a need to access data distributed among multiple databases, sometimes referenced as "federated databases," and especially where databases from multiple vendors are mixed. Years of database query optimization techniques, developed for single databases, must be reproduced through custom programming to ensure reasonable performance. So, queries across federated databases are often out of necessity rather than by design. Such queries usually span databases in a way that was initially considered unlikely, but changing business needs force the data to be looked at differently. If such queries become a regular and troublesome occurrence, consolidation of the databases to a single database is likely a good topic to consider.

Accessing Distributed Databases

Users sometimes need to query or manipulate data that resides in multiple Oracle databases, or data in a mixture of Oracle and non-Oracle databases. This section describes a number of techniques and architectures you can use to interact with data in a distributed environment.

Distributed Data Across Multiple Oracle Databases

For many years, Oracle has offered access to distributed data residing on multiple Oracle Database servers on multiple systems, or sometimes distributed among various *nodes*. Users need not know the location of the data in distributed Oracle databases. Data is accessed using a unique identifier to a specific table name. Administrators can create simple identifiers so that data in an Oracle table in a separate machine can appear to users to be part of a single logical database.

Developers can create connections between individual databases by creating Oracle database links in SQL. These connections form a distributed database. The statement:

```
CREATE PUBLIC DATABASE LINK employees.northpole.bigtoyco.com
```

creates a path to a remote database table with Bigtoyco's North Pole employees. Any application or user attached to a local employees database can access the remote North Pole database by using the global access name (`employees.northpole.bigtoyco.com`) in SQL queries, inserts, updates, deletions, and other statements. Oracle Net handles the interaction with any network protocols used to communicate with the remote database transparently.

Although the Oracle database link makes data access transparent to users, Oracle still has to treat interactions over distributed databases differently. When using distributed data in a query, the primary concern is to properly optimize the retrieval of data for a query. Queries in a single Oracle database are optimized for performance by the cost-based optimizer. Oracle can also perform global cost-based optimization for improvement of query performance across distributed databases. For example, the cost-based optimizer considers indexes on remote databases when choosing a plan, statistics on the remote databases, and optimizes for join and set operations to be performed, minimizing the amount of data sent between systems.

When a user wants to write data back to a distributed database, the solution becomes a bit more complicated. As we've mentioned before, a transaction is an atomic logical unit of work that typically contains one or more SQL statements. These statements write data to a database and must either be committed or rolled back as a unit. Distributed transactions can take place across multiple database servers. When distributed transactions are committed via the SQL COMMIT statement, Oracle uses a two-phase

commit protocol to ensure transaction integrity and consistency across multiple systems. This protocol is further described as we describe two-phase commit below.

Access to and from Non-Oracle Databases

Oracle's Gateways (illustrated in Figure 13-1) are Oracle software products that provide users with access to non-Oracle databases via Oracle SQL. Oracle SQL is automatically translated into the SQL of the target database, allowing applications developed for Oracle to be used against non-Oracle databases. You can also use native SQL syntax for the target database, which can be sent directly to the target without translation. Oracle datatypes such as NUMBER, CHAR, and DATE are converted into the datatypes of the target. Oracle data dictionary views are provided for target data store objects. Heterogeneous databases can also be accessed via Oracle database links to create a distributed database.

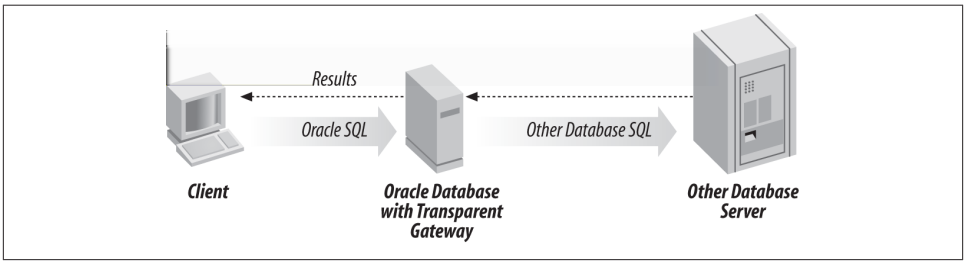


Figure 13-1. Typical configuration and use of Oracle Gateways

Key heterogeneous database connectivity support and options include:

Heterogeneous Services
: Heterogeneous Services are included in the Oracle Database and determine optimal SQL for accessing remote databases. The Services work in tandem with Oracle Gateways.

Open Database Connectivity
: Generic ODBC and OLE DB agents are commonly available for non-Oracle databases and are provided for the Oracle database by Heterogeneous Services.

Oracle Gateways
: Oracle Gateways exist for non-Oracle data stores such as (IBM) Informix, Microsoft SQL Server, (SAP) Sybase, and Teradata. The DRDA Gateway provides access to IBM data stores via IBM Distributed Relational Database Architecture (DRDA) connections.

Procedural Gateways

Procedural Gateways implement remote procedure calls (RPCs) to applications built on non-Oracle data stores. The Gateway for APPC, the standard IBM protocol for RPCs, is used when Oracle applications need procedural access to applications built on CICS, DB2, IMS, VSAM, and other data stores on the mainframe and applications that use SNA LU6.2 to communicate to the mainframe. The Oracle Procedural Gateway for IBM WebSphere MQ allows Oracle-based applications to exchange messages with applications that communicate via IBM MQ (message queues).

Two-Phase Commit

One of the biggest issues associated with the use of distributed databases is the difficulty of guaranteeing the same level of data integrity for updates to distributed databases. Because a transaction that writes data to multiple databases must depend on a network for the transmission of information, it is inherently more susceptible to lost information than a single Oracle instance on a single machine. And since a transaction must guarantee that all writes occur, this increased instability could adversely affect data integrity.

The standard solution for this problem is to use two message-passing phases as part of a transaction commit; hence, the protocol used is referred to as a *two-phase commit*. The initiating database serves as a global coordinator and first polls each of the participants to determine if they are ready and then sends transactional updates to them. In the second phase, if all the participants are in agreement that the updates have properly occurred, the changes are committed. If any of the nodes involved in the transaction cannot verify receipt of the changes, the transactions are rolled back to their original state on all the nodes.

For example, if a transaction is to span databases A, B, and C, in the first phase of the commit operation, each of the databases is sent the appropriate transactional update. If each of these databases acknowledges that it has received the update, the second phase of the update executes the `COMMIT` command. By separating the transmission of the data for the update from the actual `COMMIT` operation, a two-phase commit greatly decreases the possibility of distributed data losing its integrity.

You can compare this approach to a single-phase update in which the `COMMIT` command is sent along with the transactional update information. There is no way of knowing whether the update ever reached all the databases, so any sort of interruption in the delivery of the update to any of the machines would cause the data to be in an inconsistent state. When a transaction involves more than one database, the possibility of the loss of an update to one of the databases increases greatly; that, in turn, mandates the use of the two-phase commit protocol. Of course, since the two-phase commit protocol requires more messaging to be passed between databases, a two-phase commit can take

longer than a standard commit; however, the corresponding gain in all-important data integrity more than makes up for the decrease in performance.

In 1991, The Open Group defined an open systems standard interface known as X/Open eXtended Architecture (XA) for executing transactions including two-phase commit. XA-compliant transaction processing (TP) monitors communicate with XA resources, such as the Oracle database. Early examples of popular TP monitors that provided XA supported included Tuxedo, IBM's CICS, and Encina. When Oracle acquired BEA in 2008, Tuxedo joined Oracle's family of products.

Oracle Tuxedo

Where Oracle Tuxedo is deployed, distributed transaction applications can be developed in a variety of programming languages. Tuxedo can act as an ATMI (Applications to Model Interface) server where client applications written in C, C++, or COBOL can be deployed to make requests. Java ATMI (JATMI) extends the usefulness of these applications to Java programs. The programming model supports synchronous calls, asynchronous calls, nested calls, forwarded calls, conversational communication, unsolicited notification, event-based communication, queue-based communication, and using transactions for two-phase commit.

A Service Component Architecture (SCA) style of programming is often used. SCA composites are written in an XML-based format called the Service Component Definition Language that describes the components. SCA components can be configured as ATMI and JATMI clients, as workstation clients, as hosted in Tuxedo servers, or as web-service servers. Client security is offered through Tuxedo Application Domain Security and through Tuxedo Link-Level Security. Web service-based applications written in Python, Ruby, or PHP can be integrated with Tuxedo using the Oracle SALT (Service Architecture Leveraging Tuxedo). Coding using these languages can be advantageous compared to coding in C/C++ since no compilation is required and dynamic data typing takes place.

CORBA applications development is another programming paradigm sometimes used, though Oracle's CORBA Tuxedo Java Client and Tuxedo CORBA Java client ORB were deprecated as of Tuxedo 8.1. Tuxedo CORBA is limited to support for third-party object request brokers that implement standard IIOP.

A number of Oracle Tuxedo integration capabilities exist beyond those already mentioned. Oracle Tuxedo Jolt, a Java class library and API, enables the Oracle WebLogic Server to invoke ATMI services and provide Java client access to ATMI. Tuxedo .NET client support is offered. The Oracle Application Rehosting Workbench enables z/OS CICS applications to run unchanged through API emulation on Oracle Tuxedo. The MQ Adapter enables bi-directional connectivity to IBM WebSphere MQ queues.

Today, there are alternative means to assure reliable delivery of distributed transactions, a capability once available only with a TP monitor. In Chapter 9, we cover many of these capabilities provided by the Oracle database. Standalone TP monitors are also used less frequently today for workload management (see Figure 13-2) as applications leverage other features in middle-tier applications servers and clustered databases to better distribute the workload.

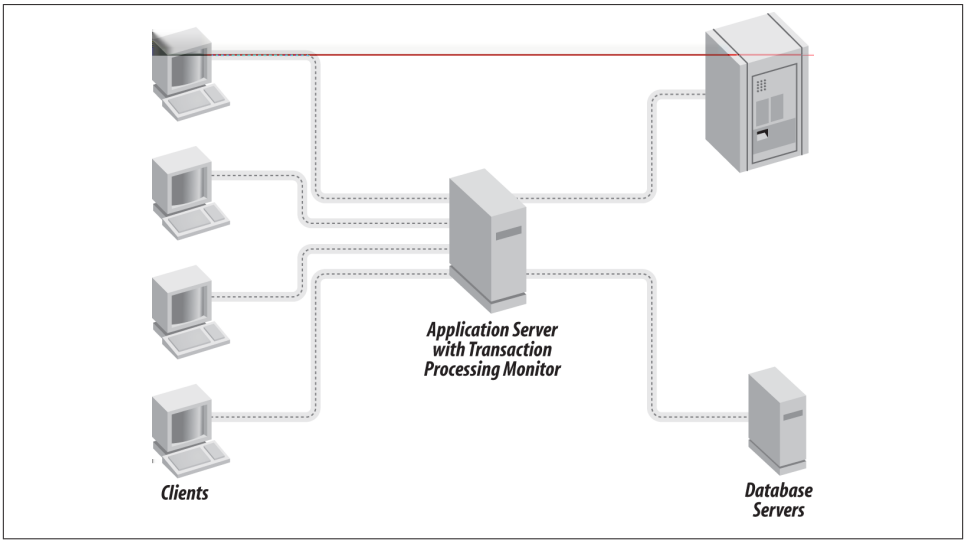


Figure 13-2. Application server with TP monitor

If you are still considering the use of TP monitors, you likely have one of these scenarios:

- Migration of legacy applications (usually originally written using CICS and COBOL for a mainframe) to Unix or Windows-based platforms
- Migration of older Tuxedo-based applications to a more modern footprint and current Tuxedo version
- Need for two-phase commits between Oracle and other XA-compliant databases

Next we will look at replication and how it has changed over the years in Oracle's products. These approaches present one popular alternative to programming with TP monitors today.

Replication and Data Transport

The previous section discussed the use of multiple database servers acting together as part of a single logical database for users. In this section, we cover data replication and data transport in order to duplicate data.

Replication techniques are frequently used:

- When data available locally eliminates network bandwidth issues or contention for system resources
- When mobile database users can take their databases with them and operate disconnected from the network
- When redundant databases can help to deliver higher levels of reliability, as each database can be used as a backup for other databases
- During a database or application migration

In grid implementations, the ability to share resources across the grid of computer systems and databases can also require data to be replicated to multiple servers within the grid.

The biggest issue facing users of multiple identical or similar databases is how to keep the data on all of the servers in sync as the data is changed over time. As a user inserts, updates, or deletes data on one database, you need to have some way to get this new data to the other databases. In addition, you will have to deal with the possible data-integrity issues that can crop up if the changes introduced by distributed users contend with each other.

Oracle offered a number of strategies to address this situation over the years, including Advanced Replication in the Oracle Database, Oracle Advanced Queuing (AQ), and Streams. In Oracle Database 12*c*, some of Oracle's features still leverage AQ and Streams under the covers. However, a number of years ago, Oracle began guiding its customers toward using Oracle GoldenGate for asynchronous replication and Data Guard for synchronous replication.

Replication Basics

The copying and maintaining of database tables among multiple Oracle Databases on distributed systems is known as *replication*. Changes that are applied at any local site are propagated automatically to all of the remote sites. These changes can include updates to data or changes to the database schema. Replication is frequently implemented to provide faster access for local users at remote sites or to provide a disaster-recovery site in the event of loss of a primary site.

A master database logs changes that are then applied to one or more targets. *Synchronous replication* is a means to guarantee zero data loss between the master and target database site since the replication is not considered complete until the target database is updated and acknowledges the update. *Asynchronous replication* is considered complete when the master is ready to forward the changes to the target, so there is usually a time lag between the master and the target being identical. Examples of asynchronous replication include read-only snapshots replicated from a single updateable master table to a target and disconnected updateable snapshots.

Multimaster replication introduces master groups where changes are logged at multiple locations. A more complex but potentially more highly available replication, multimaster replication solutions can handle conflict resolution scenarios to ensure consistency.

History of Oracle Replication Offerings

Oracle initially featured replication services as part of the Oracle database, and moved execution of replication triggers to the database kernel, enabling automatic parallelization of data replication to improve performance. Advanced Replication featured support for both asynchronous replication and synchronous replication and was capable of replication to non-Oracle databases through gateways. Multimaster replication was supported. Administrators configured database objects that needed to be replicated, scheduled replication, troubleshot error conditions, and viewed deferred transaction queues through Oracle Enterprise Manager. As of Oracle Database 12*c*, Advanced Replication was fully deprecated as an Oracle-supported feature, as its capabilities have been replaced by Oracle GoldenGate.

Message-oriented solutions gained popular usage since using *messages* to transmit information between systems doesn't require the overhead of a two-phase commit. Control information (message destination, expiration, priority, and recipients) and the message contents are placed in a file-based queue. Delivery is guaranteed in that the message will remain in the queue until the destination is available and the message is forwarded.

Oracle's Advanced Queuing (AQ) facility was introduced with Oracle8 Enterprise Edition and later became a part of Oracle Streams. The queues are stored in Oracle Database tables and the database enables queuing operations—in particular, *enqueue* to create messages and *dequeue* to consume them. These messages, which can either be unstructured (raw) or structured (as Oracle objects), correspond to rows in a table. Messages are stored in normal queues during normal message handling or in exception queues if they cannot be retrieved for some reason.

AQ also has publish-and-subscribe features for additional notification of database events that, in turn, improve the management of the database or business applications. Database events such as DML (inserts, updates, deletions) and system events (startup, shutdown, and so on) can be published and subscribed to. As an example, an application

could be built to automatically inform a subscriber when a shipment occurs to certain highly valued customers; the subscriber would then know that she should begin to track the shipment's progress and alert the customer that it is in transit.

Oracle9*i* Release 2 introduced Oracle Streams, which folded the capabilities of Advanced Replication and AQ into a single-product family and added a method of sharing data and events within a database or between databases. Streams enable the propagation of changes via a capture-and-apply process, including Oracle's change data capture. Changes can be propagated between Oracle instances, from Oracle instances to non-Oracle instances (via Oracle Gateways), and from non-Oracle databases to Oracle (via messaging gateways in combination with custom code on the non-Oracle source to collect changes). Streams leverages log-based procedures to capture DML or DDL changes or synchronous capture for DML changes and then uses queuing procedures as part of the staging. User-supplied "apply" rules define consumption.

When changes are captured from an Oracle database redo log or changes in rows are gathered from synchronous capture, a background database process creates a logical change record (LCR). LCR and user message events are enqueued in a Streams queue. Events are propagated from source to target queues and then, via a background process, dequeued in the target database and applied. Since Oracle Database 10*g*, downstream capture of changes and enqueue/dequeue of messages in batch are supported, and Streams can be configured to provide Database Change Notification via email, HTTP, and PL/SQL. This feature can send notifications to a client whenever the data in a query result set has changed.

Oracle GoldenGate

Shortly after Oracle acquired GoldenGate in 2007, Oracle began to provide guidance that it would become part of Oracle's data integration middleware solution (Oracle Data Integrator Suite) and that organizations using Advanced Replication, AQ, and Streams should evaluate it as part of their go-forward strategy. Oracle then began adding features and functions present in earlier replication solutions that had been missing in GoldenGate. Today, GoldenGate supports replication to and from databases from multiple vendors including Oracle databases, Oracle TimesTen, Oracle MySQL, Microsoft SQL Server, IBM DB2, and others.

Data can be replicated using GoldenGate from a single source to a single target, a single source to many targets, many sources to a single target, many sources to many targets, and using cascading and bi-directional deployment models. GoldenGate provides conflict resolution in multimaster configurations where two systems can modify different instances of the same table.

Oracle GoldenGate is often used to provide near real-time updates. It can capture, route, transform, and deliver transactional data to other systems with subsecond latency (especially where minimal data transformations are required). It maintains atomicity,

consistency, isolation, and durability (ACID) properties as transactions are moved between systems and ensures data consistency and referential integrity across multiple masters, backup systems, and reporting databases.

GoldenGate can be deployed in a variety of scenarios in addition to replication among transactional databases. As part of a data warehouse infrastructure, it can be integrated with Oracle Data Integrator Enterprise Edition for ETL and changed data can target staging tables.

Key components in Oracle GoldenGate include the Capture mechanism, Trail Files, the Delivery mechanism, and GoldenGate Manager. "Classic Capture" relies on access to transaction logs (in the case of Oracle or other databases such as Microsoft SQL Server, IBM DB2, and Sybase) or APIs (such as with Teradata). This Capture mode supports checkpoints, enabling restarts from the last good checkpoint. In "Integrated Capture Mode," GoldenGate provides integration between GoldenGate Extract and the Oracle log mining server, enabling data to be received as Logical Change Records (LCRs).

Trail Files consist of changed data in the GoldenGate Universal Data Format. Trail Files exist on the source and are most often pumped to the target(s) using the GoldenGate Data Pump capabilities. Most often, data is transported using TCP/IP, though the Java Message Service (JMS) is also supported. Data can be compressed and encrypted (variable key length) during transportation.

On the Delivery side, transactional data is applied to the target database(s). This is where changed data is published to JMS or pushed to ETL tools. Changes can be applied immediately or on a deferred basis. Bounded recovery is supported.

The GoldenGate Manager provides a command line interface to set GoldenGate parameters, the capability to start, stop, and monitor the GoldenGate Capture and Delivery modules, event and threshold reporting, resource management, and Trail File management. The Management Pack for Oracle GoldenGate provides a monitor and plug-in for Oracle Enterprise Manager.

Oracle GoldenGate Veridata is an option for GoldenGate and is used to detect data discrepancies between databases. The databases might include Oracle and non-Oracle databases such as Teradata and Microsoft SQL Server. Administrators can pick and choose what data they want to compare between the various databases.

GoldenGate is paired with Data Guard for replication in high availability implementations where zero data loss is required. We covered high availability and introduced Data Guard and this concept in Chapter 11.

Global Data Services

Oracle *Global Data Services* (GDS), introduced in Oracle Database 12*c*, are used to integrate multiple replicated Oracle databases into a common configuration accessed

by global clients. The GDS configuration acts as a single virtual server supporting one or more common workloads as global services. Optimal workload resource utilization is managed across the replicated Oracle databases. Key components in the architecture include Active Data Guard, Data Guard Broker, and GoldenGate.

GDS Oracle Databases that are managed by a common administrator are said to be in the same GDS pool. All Oracle databases providing the same global service(s) must be in the same pool.

GDS regions are typically geographically bounded. The Global Service Manager acts as a regional listener that clients use to connect to the global services they need. The Global Service Manager also provides service-level load balancing, failover, and centralized management of services. The metadata repository that stores GDS configuration and global service data is called the GDS catalog.

Data Transport Using Database Features

The previous sections focused on sharing data between distributed databases when the data is "live" by propagating changes among databases. Oracle also provides ways to speed up the distribution of data through the export and import of tablespaces, tables, or entire databases.

Transportable tablespaces are a way to speed up the distribution of complete tablespaces between multiple databases while the tablespaces are not active. They were introduced with Oracle8*i* Enterprise Edition to rapidly copy and distribute tablespaces among database instances. Previously, tablespaces needed to be exported from the source database and imported at the target (or unloaded and loaded). Transportable tablespaces enable copies to be moved simply through the use of file transfer commands such as *ftp*.

Transportable tablespaces have long been popular in data warehousing for copying from data warehouses to data marts. They've also been useful when used as backup copies for rapid point-in-time tablespace recovery.

Before you copy and move a copy of a tablespace, make sure the tablespace is read-only to avoid inadvertently changing it. Data dictionary information needs to be exported from the source prior to transfer, and then imported at the target. Cross-platform backups and restores using transportable tablespaces can be performed where Oracle Database 12*c* is deployed, somewhat simplifying data movement process across platforms.

Where transportable tables are used, they automatically identify the tablespaces used by the tables. In fact, you can use this feature to copy tables, partitions, or subpartitions from one database to another. When moving transportable tablespaces and tables among current Oracle Database releases, you must use the Oracle Data Pump (instead of the legacy import and export features of the Oracle database used prior to Oracle Database 10*g*).

There are situations where you want to do a full transportable export and import of an Oracle database. You might do this when moving an Oracle database from an older computer system being retired to a new computer system. You might also use this capability during the upgrade to a newer Oracle database release (where the older release of Oracle is Oracle Database 11*g* Release 2 or later).

Similarly, where Oracle Database 12*c* is deployed, you might move an entire pluggable database (PDB) in an Oracle multitenant container database (CDB) into another PDB. You can also use Data Pump full transportable capabilities to move an Oracle database that is not a CDB (from Oracle Database 11*g* Release 2 or later) into an Oracle PDB.

If your replication needs allow it, you can also do a partition exchange, where a single partition of a table is moved to a target machine and then rapidly added to the table. This feature has similar characteristics to using transportable tablespaces, but typically requires a smaller amount of data, which reduces transmission times to the target database.

CHAPTER 14
Oracle Extended Datatypes

You might find that your data is diverse and extends beyond types of data typically found in most relational databases. Specialty databases, such as object databases and XML databases, emerged at various times to address these needs. As noted in Chapter 10, Hadoop and the Hadoop Distributed File System (HDFS) are popular as a data store and engine for unstructured and semi-structured data today.

Where most of the data to be processed is structured, it can make sense to do the processing of all of the data in the relational database. Earlier in Chapter 4, we covered the rich set of native datatypes in the Oracle database with a focus on what is required in traditional relational databases. Oracle also provides datatypes that are specifically designed to provide optimal storage, performance, and flexibility for other types of data —the focus of this chapter.

For example, object datatypes in Oracle can be used to represent purchase orders, claims forms, shipping forms, and so on in a single unified entity. The XML datatype and support for features such as XMLSchema, an XML DB repository (enabling URL-based access to XML documents stored in Oracle), and SQL/XML (for generating XML documents from SQL) extend Oracle's ability to blend the relational database with characteristics of an XML database. Location-oriented data may best be represented using spatial coordinates stored in the Oracle database. Documents, images, video clips, and audio clips have their own special requirements for storage and retrieval and Oracle supports those as well.

Oracle extended the functionality of its basic relational database engine to support the storage and manipulation of these nontraditional datatypes through the introduction of additional features and options. By taking advantage of the extended types of data, extended SQL that manipulates that data, and the Oracle Extensibility Architecture framework, you will find even greater flexibility in how you might deploy and use the Oracle database.

Object-Oriented Development

An object-oriented approach to software development shifts focus from building computing procedures that operate on sets of data to modeling business processes. Building software components that model business processes with documented interfaces makes programming more efficient and allows applications to offer more flexible deployment strategies. It also makes applications easier to modify when business conditions change. In addition, since the modeling reflects real business use, application performance may improve as objects are built that do not require excessive manipulation to conform to the real-world behavior of the business processes they represent.

Oracle took an evolutionary approach to object technology by allowing *data abstraction*, or the creation of user-defined datatypes as objects and collections as extensions to the Oracle relational database. The objects and extensibility features date back to the late 1990s in Oracle8*i*, enabling Oracle to provide object-relational capabilities.

The Java language support in the Oracle database complements this approach. The JVM feature is a Java Virtual Machine integrated with Oracle. It supports the building and running of Java components, as well as Java stored procedures and triggers, in the server.

The Promise of Code Reuse

Although a number of object-oriented approaches and technologies have been introduced over several decades, many of the promised improvements in software development efficiency have largely been unrealized. One of the reasons that these productivity improvements have failed is the difficulty many developers have had in making the adjustment to building reusable components. In addition, the need to learn new languages (such as C++) and technologies (object-oriented databases, CORBA, DCOM, and .NET) slowed the adoption of object-oriented development. Developers did become more familiar with these techniques and skills as Java moved into the mainstream of development. Interestingly, Oracle leverages many of these object capabilities in the Oracle database in development of new extensions.

Object-Relational Features

This section describes the major object-relational features available in Oracle.

Objects in Oracle

Objects created in Oracle are reusable components representing real-world business processes. The objects created using the database objects and extensibility features occupy the same role as the table in a standard relational model: the object is a template for the creation of individual "instances" of the object, which take the same role as rows

within a table. An object is "instantiated" using Oracle-supplied "constructors" in SQL or PL/SQL.

An *object* consists of a name, one or more attributes, and methods. *Attributes* model the structure and state of the real-world entity, while *methods* model the operations of the entity. Methods are functions or procedures, usually written either in PL/SQL or Java or externally in a language such as C. They provide an interface between an object and the outside programming environment. Each method is identified by the name of the object that contains the method and a method name. Each method can have one or more *parameters*, which are the vehicles for passing data to the method from the calling application.

For example, a purchase order can be represented as an object. Attributes can include a purchase order number, a vendor, a vendor address, a ship-to address, and a collection of items (with their associated quantity and price). You can use a method to add an item to the purchase order, delete an item from the purchase order, or return the total amount of the purchase order.

You can store objects as rows in tables or as values in columns. Each row object has a unique object identifier (OID) created by Oracle. Row objects can be referred to from other objects or relational tables. The REF datatype represents such references. For column objects, Oracle adds hidden columns for the object's attributes.

Object views provide a means of creating virtual object tables from data stored in the columns of relational tables in the database. These views can also include attributes from other objects. Object views are created by defining an object type, writing a query defining the mapping between data and tables containing attributes for that type, and specifying a unique object identifier. When the data is stored in a relational table, the unique identifier is usually the primary key. This implementation means that you can use object-oriented programming techniques without converting existing relational tables to object-relational tables. The trade-off when using this approach is that performance may be less than optimal, since the data representing attributes for an object may reside in several different tables. Hence, it may make sense to convert the relational tables to object tables in the future.

Objects that share the same attributes and methods are said to be in the same datatype or *class*. For example, internal and external purchase orders can be in the same class as purchase orders. *Collection types* model a number of objects of the same datatype as varying arrays (VARRAYs) if the collection of objects is bounded and ordered or as nested tables if the collection is unbounded and unordered. If a collection has fewer than 4,000 bytes, it is stored in a VARRAY as part of the database table column as a raw value; if it is larger, it is stored as a Binary Large Object (BLOB) in a segment separate from the table that is considered "out-of-line" storage. Nested table rows are stored in a separate table identified through a hidden NESTED_TABLE_ID by Oracle. Typically, VARRAYs are used when an entire collection is being retrieved and nested tables are

used when a collection is being queried, particularly if the collection is large and only a subset is needed.

An application can call object methods through SQL, PL/SQL, Pro*C/C++, Java, OCI, and the Oracle Type Translator (OTT). The OTT provides client-side mappings to object types by generating header files containing C structure declarations and indicators. Developers can tune applications by using a client-side object cache to improve performance.

Inheritance, or the use of one class of objects as the basis for another, more specific class, is one of the most powerful features of object-oriented design. The child class inherits all the methods and attributes of the parent class and also adds its own methods and attributes to supplement the capabilities of the parent class. The great power of inheritance is that a change in a parent class automatically ripples down to the child classes. Object-oriented design supports inheritance over many levels of parent, child, and grandchild classes.

Polymorphism allows handling of multiple datatypes and methods through a common interface. Polymorphism overriding describes the ability of a child class to supersede or "override" the operation of a parent method by redefining the method on its own. Once a method has been replaced in a child class, subsequent changes to the method in the parent class don't ripple down to the child class or its descendants. In the purchase order example, as shown in Figure 14-1, purchase orders from contracted suppliers and suppliers not under contract inherit the methods and attributes of external purchase orders. However, the procedure for placing the order can exhibit polymorphism because additional approvals may be required for ordering from suppliers not under contract.

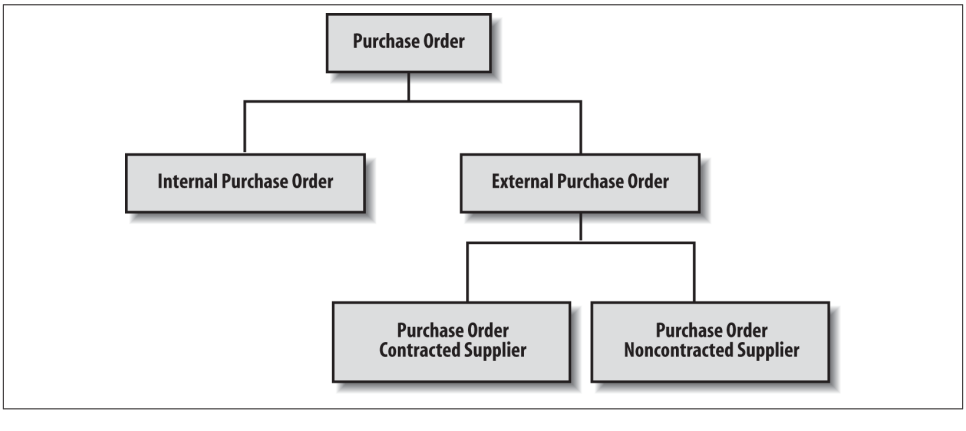


Figure 14-1. Purchase order class hierarchy

Inheritance and polymorphism were not supported in Oracle8*i* objects, though the Oracle8*i* database could act as persistent storage for objects, and an application interface

in an object-oriented language such as C++ or Java could add these features to the client-side implementation of objects. Oracle9*i* added SQL-type inheritance to the database, as well as object view hierarchies, type evolution, generic and transient datatypes, function-based indexes for type method functions, and multilevel collections. Oracle Database 10*g* added support for remote access to object types and Oracle Database 11*g* added an ANSI SQL feature that provided a method invocation scoping operator.

Other extensibility features

Several other extensibility features are included among the objects and extensibility features. These include:

- The ability to create new indexes on object tables
- The ability to store the index data inside or outside the Oracle Database
- The ability to create user-defined constructor functions for creating and initializing user-defined types
- The ability to create user-defined aggregate functions for use in standard SQL statements
- An interface to the cost-based optimizer to extend support for user-defined object types and indexes

The use of object-relational features is most common today among software developers who are building database extensions. As we noted earlier, Oracle itself has made use of these features in the creation of many of the database features—for example, in the spatial and multimedia capabilities. These capabilities are discussed in more depth later in this chapter.

Java's Role and Web Services

Java has gained wide acceptance as an application language due to its portability and availability on a wide variety of platforms.

For Java developers wanting to use the Oracle database as a backend to their applications, Oracle offers support for the two common approaches for accessing it from a Java program: JDBC and SQLJ. Both of these approaches are based on industry-standard application programming interfaces (APIs):

JDBC
: More commonly used since it can be used where SQL is dynamic, or when a developer wants explicit control over interactions with the database.

SQLJ
: Used when static SQL statements are embedded into a Java program. SQLJ is similar to other Oracle precompilers in that Java source files are created with calls to the

SQLJ runtime (as well as to additional profile files). The Java source code is then compiled, and the application is run with the SQLJ runtime library.

SQLJ and JDBC can be mixed in the same program when some SQL is static and other SQL is dynamic.

Oracle features a tightly integrated Java Virtual Machine and support for Java stored procedures in the Oracle database; these enable component-based development to take place through the use of JavaBeans. Java Messaging Support (JMS) is provided through Oracle Streams when deployed in the database.

The Oracle database can act as a Web Services consumer or provider and can be exposed using JPublisher, Oracle's utility for generating Java classes that represent user-defined database entities. Web services capabilities in the Oracle database include SQL, PL/SQL, embedded Java, JDBC, HTTP client, and SOAP client, and are combined with those in Oracle Application Server (Java, J2EE, JDBC, HTTP, SOAP server, and XML). The latest releases of the APEX Listener and Oracle Application Express also support the easy creation and use of RESTful Web Services, which are described in more detail in Chapter 15.

As of Oracle Database 11*g*, the database can be treated as a service provider in a service-oriented architecture (SOA) environment using the XDB HTTP Server for SOA. PL/SQL packages, procedures, and functions can be exposed as Web Services. Dynamic SQL and XQuery queries can be executed when deploying the database in this manner.

JavaBeans

Java software components are referred to as JavaBeans (and sometimes referred to as Enterprise JavaBeans or EJBs when server-side). JavaBeans can be deployed in the Oracle database server or in the Oracle Fusion Middleware application server. The Java Virtual Machine in the database makes use of Oracle System Global Area (SGA) memory management capabilities to provide JavaBeans scalability beyond what would be expected in most JVM implementations. For example, each client within the JVM requires only about 50–150 KB of memory for session state.

Oracle8*i* introduced the *session bean* to the database, created by a specific call from the client and usually existing only during a single client/server session. Session beans may be *stateless*, allowing the server to reuse instances of the bean to service clients, or *stateful* (i.e., bound to clients directly). Database cache information maintained by stateful session beans is synchronized with the database when transactions occur by using JDBC or SQLJ. *Entity Java beans*, also known as *persistent beans* (because they remained in existence through multiple sessions), were introduced in Oracle9*i* but were later replaced by Java persistence API entities. Today, the other type of bean available is the *message-driven bean*, designed to receive asynchronous Java Message Services (JMS) messages and supported via Oracle's more recent Applications Servers.

Extensibility Features and Options

Oracle's extensibility features and options extend SQL and storage to perform tasks that can't otherwise easily take place in relational databases, including manipulation of multimedia, text, XML data, and spatial and graph data. These features are often used by application developers. Some are key enablers to applications sold by Oracle and its partners.

Oracle Multimedia

Oracle Multimedia, formerly known as *inter*Media, is included with all editions of the Oracle database with the exception of the Express Edition. It is used to store, manage, and retrieve multimedia data including:

Audio data
: Media data produced by an audio recorder or audio source are supported in the following audio file and compression formats: 3GP, AIFF, AIFF-C, AU, MPEG, Real Networks Real Audio (RMFF), WAV, and Windows Media ASF

DICOM data
: Medical image information objects encoded according to DICOM standards with methods and functions to copy and process this content into DICOM (Oracle Database 12*c* adds support for the DICOM communications protocol used in the exchange of DICOM images) and other image formats (JPEG, GIF, PNG, TIFF) and video formats

Image data
: Data produced by scanners, video sources, other image capture devices, or programs producing image formats are supported in the following image file formats: BMPF, CALS, FPIX, GIFF, JFIF, PBMF, PCXF, PGMF, PICT, PNGF, PNMF, PPMF, RPIX, RASF, TGAF, TIFF, and WBMP; and supported in the following image compression formats: none, JPEG, JPEG-PROGRESSIVE, BMPRLE, PCXRLE, SUNRLE, TARGARLE, GIFLZW, GIFLZW-INTERLACED, LZW, LZWHDIFF, FAX3, FAX4, HUFFMAN3, PACKBITS, DEFLATE, DEFLATE ADAM7, ASCII, and RAW

Video data
: Data produced by video recorders, video cameras, digitized animation, or other video devices or programs producing video formats are supported in the following formats: Apple QuickTime 3.0, Microsoft Video for Windows (AVI), Real Networks Real Video (RMFF), 3GP, Video MPEG, and Windows Media File Format (ASF)

Heterogeneous data
: Assorted audio, image, video, and other data in a variety of formats such as those previously listed

The JVM for Multimedia in the Oracle database provides a server media parser and an image processor. Five object relational types, collectively known as ORDSource, store data source information: ORDAudio (for audio multimedia), ORDDoc (for heterogeneous multimedia), ORDImage (for image multimedia), ORDVideo (for video multimedia), and ORDDicom (for DICOM image multimedia). In Oracle Database 12*c*, Multimedia can also be deployed to pluggable databases (PDBs). Oracle Fusion Middleware provides access to Multimedia through Oracle Java Multimedia classes.

Multimedia files are typically loaded into the Oracle database using SQL*Loader or PL/SQL scripts. They are stored as Multimedia object relational types or directly in BLOBs or BFILEs. Applications access multimedia data via the Multimedia Java API. You might also stream content via plug-ins, integrate multimedia into web applications using the Multimedia Servlets and JSP Java API class library, or use the Java Advanced Imaging (JAI) classes to interface to BFILE and BLOB data. Other Multimedia Java class libraries that are used in building applications include the Oracle Multimedia JSP tag library, DICOM Java API class library, and Mid-Tier Java API Class library.

Oracle Text

Oracle Text enables document and catalog retrieval and viewing, indexing, searching for words, and theme determination and classification. Text is included with all editions of the Oracle database. Documents accessed through Oracle Text might be stored in the Oracle database, in file systems, or on the Web. The Text SQL API supports creation and maintenance of Oracle Text indexes and enables searches that can be combined with other database searches. You can also manage the indexes through the Text Manager in Oracle Enterprise Manager.

Queries submitted that include text in the Oracle Database are optimized by Oracle's cost-based optimizer. You can also use the CONTEXT index type to build web query applications using PL/SQL Server Pages (PSPs) or Java Server Pages (JSPs).

Typically, text that is searched includes HTML tagged documents or XML documents. You can also perform name searching and matching. Thesauruses can be created, modified, deleted, imported, and exported using Oracle Text.

Oracle Text supports the building of document classification applications that will perform an action based on content in the documents. Classification applications can be rule-based (where you formulate rules that define the classifications), supervised (using automated rule writing functions), and unsupervised automated clustering.

XML DB

XML DB is a term for a group of XML features that were first introduced as part of Release 2 of Oracle9*i* and is included with all editions of the Oracle database. XML DB provides support for the World Wide Web Consortium (W3C) XML Schema used to

specify the structure, content, and semantics of an XML document. As of Oracle Database 12*c*, XML DB is a mandatory part of the Oracle database installation. XML DB and the XMLType abstract datatype are what combine to make the Oracle database XML-aware. XML processing performance in Oracle is improved by XML storage optimization, reduced memory overhead, reduced parsing, optimized node searching, XML schema optimization, and load balancing through cached XML schema.

The XML DB Repository enables organizing of XML data via files and folders (directories, containers) in a hierarchy and provides a model of traversing paths and URLs when accessing this data. Oracle Database 12*c* extended access from the XML DB Repository into the Oracle Database File System (DBFS) files and folders. The Repository secures access and manipulation control is through the use of Access Control Lists (ACLs). Repository documents can be accessed via HTTP(S), WebDAV and FTP, and SQL via Oracle Net Services including JDBC. XML messaging is supported thorough Web Services and Oracle Streams Advanced Queuing (AQ).

The XMLType datatype indicates to the Oracle database that the data is XML data, so that specific XML data operations, such as those initiated by XQuery or are XPath-based, can be performed. The default storage type in the Database for XMLType is binary XML storage (since Oracle Database 11*g* Release 2) and is stored using large objects (LOBs). XMLIndex is used to index data stored in binary XML.

XQuery is the W3C language used in querying and updating XML data. XPath is a subset of the XQuery language. Oracle Database 12*c* added the XQuery update capability as well as XQuery Full Text and the XQuery API for Java (XQJ).

XML DB also provides the SQL functions that are defined by the SQL/XML standard. These functions provide the capability to generate or publish XML data from the result of a SQL query and also query and access XML data via SQL. They use the XQuery or XPath functions to search XML documents and access a subset of an XML document.

Programmatic access to XML DB is available using Java, PL/SQL, and C. Available APIs for XMLType include the Document Object Model (DOM), XML Parser, and the XSLT Processor APIs. Web-based applications can be built using servlets and Java Server Pages (JSPs), and using the Extensible Stylesheet Language (XSL) and XML Server Pages (XSPs).

Oracle Spatial and Graph Option

The Spatial and Graph Option (formerly Oracle Spatial Option) for the Oracle Database Enterprise Edition provides the following:

Spatial Data Extended Support
: Spatial functions and GeoRaster types useful for defining topology data maps and building advanced geographic information system (GIS) and location-based services applications

Network Data Model Graph
: Support for modeling and analyzing link-node graphs in the database useful in transportation modeling and similar applications

Semantic Database Management
: Support for W3C Resource Description Framework (RDF) semantic graphs and Web Ontology Language (OWL)

Spatial data can be most simply defined as data that contains location information. Spatial data support has existed in the Oracle database for decades. The Oracle Spatial and Graph Option provides functions and procedures used to enable spatial data applications and is only available for Oracle Enterprise Edition.

Spatial queries can combine spatial and standard relational conditions, a typical example being "find all homes within two square miles of the intersection of Main Street and First Avenue in which the residents' income is greater than $100,000, and show their location." This query might return a list of home addresses or, when used with a Geographic Information System (GIS), plot the home locations on a map, as shown in Figure 14-2. Geocoding matches references such as addresses, phone numbers (including area codes), and postal codes (with longitude and latitude), which are then stored in the database.

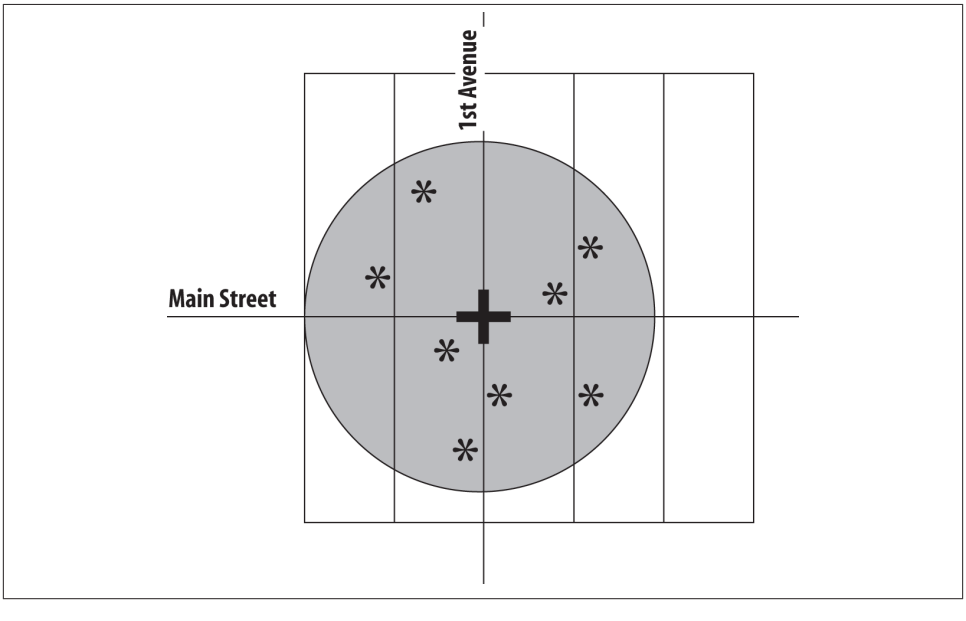


Figure 14-2. Geographic Information System display of a spatial query

Multiple geometric forms are supported by the Oracle Spatial and Graph Option to represent many different types of spatial data, including points and point clusters, lines and line strings, polygons and complex polygons with holes, arc strings, line strings, compound polygons, and circles. Oracle Database 12*c* adds support for nonuniform rational B-spline (NURBS) curve geometries that allow representation of arbitrary shapes. You can determine the interaction of these features through the use of operators such as TOUCH, OVERLAP, INSIDE, and DISJOINT.

Data that shares the same object space and coordinates but represents different characteristics (such as physical and economic information) is often modeled in layers. Each layer is divided into tiles representing smaller subareas within the larger area. A representation of this tile is stored with a spatial index that provides for quick lookups of multiple characteristics in the same tile. The Spatial and Graph Option uses these representations to rapidly retrieve data based on spatial characteristics. For example, you can perform a query against a physical area to examine where pollutants, minerals, and water are present. Each of these characteristics is likely to be stored in a separate layer, but they can be quickly mapped to their common tiles. The designers of these spatial-based databases can increase the resolution of the maps by increasing the number of tiles representing the geography.

The Spatial and Graph Option fully leverages Oracle's object features through the use of a *spatial object type* that represents single or multi-element geometries. Spatial coordinates are stored in VARRAYs.

Support for GeoRaster data has existed since Oracle Database 10*g* and enables storing, indexing, querying, analyzing, and delivering raster image data, associated spatial vector geometry data, and metadata. This feature enables storage of multidimensional grid layers and digital images in object-relational schema that are referenced to coordinate systems. Oracle Database 11*g* added three-dimensional geometry objects and enhanced Web Services support including business directory, Web Feature Service (WFS), Catalog Services for the Web (CSW), and OpenLS support. Oracle Database 12*c* adds support for raster algebraic expressions and analytics as well as support for advanced image processing. Examples of the advanced image processing now supported include advanced georeferencing, reprojection, rectification, orthorectification, raster update, raster appending, large-scale physical mosaics, virtual mosaics, and ad hoc spatial queries over virtual mosaics.

In the real world, most spatial implementations aren't custom-built from SQL, but instead utilize purchased GIS solutions that are built on top of databases. Many of these GIS providers include Oracle Spatial and Graph Option technology as part of their product bundles.

Support for the Network Data Model Graph was first added to this option in Oracle Database 11*g* for defining and modeling of nodes, links, paths, subpaths, logical networks (without geometric information), and spatial networks. Oracle Database 12*c* adds

support for modeling objects of interest as features in the Network Data Model and for multimodal (e.g., multiple modes of transportation) and temporal modeling support.

Support for semantic graph data (RDF and OWL) was also first introduced in this option for Oracle Database 11*g*. Simply put, the RDF model is a means of representing URI references as subjects, objects, and predicates for processing. OWL is a semantic markup language for publishing ontologies on the Web and is an extension of RDF. Oracle added some additional capabilities to Oracle Database 12*c* to enable user-defined rule-based inference and querying, ladder-based inference, and RDF views.

The Extensibility Architecture Framework

It is possible to extend the basic functionality of the Oracle database using Oracle's Extensibility Architecture framework. The framework provides entry points for developers to add features to the existing database feature set. Using this framework you can:

Create new relational or set operators for use in SQL statements
: These operators can be useful when working with extended datatypes, such as multimedia or spatial data. You can create relational operators that relate specifically to a particular datatype, such as the relational operator CLOSER TO, which you can use in SQL statements that access spatial data.

Create cooperative indexing
: Cooperative indexing is a scheme in which an external application is responsible for building and using an index structure that you can use with complex datatypes. The indexes created are known as *domain indexes*.

Extend the optimizer
: If you use extended indexes, user-defined datatypes, or other features, you can extend the statistics-collection process or define selectivity and cost functions for these extended features. The cost-based optimizer can then use these to choose an appropriate query plan.

Add cartridge services
: These are services used by Oracle database extensions (such as spatial), providing memory management, context management, parameter management, string and number manipulation, file I/O, internationalization, error reporting, and thread management. These services are available to software developers to provide a means to create uniform integration of extensions with the Oracle database.

When you add and extend database functionality through the extensibility framework, you can still take advantage of core database features, such as security management, backup and recovery, and SQL.

CHAPTER 15

Oracle and the Cloud

In the first four editions of this book, this final chapter was something of a catch-all. We spent the previous 14 chapters outlining and hopefully illuminating the principles and processes that worked together to produce the Oracle database. This chapter would cover a number of things that did not fit into the overall structure of the book, but were still important enough that most Oracle practitioners would want and need to understand them.

With this edition, we can finally pull most of these separate parts together as we describe the Oracle database in the cloud. This chapter will cover some important, and frequently misunderstood, basics about the different varieties of cloud computing, describe the various options for using the Oracle database in the cloud, and look at the Oracle Database Cloud and your options for building your own Oracle Database Cloud in more depth.

Cloud Definitions

No doubt about it, at the time of this writing, the cloud is mentioned everywhere—it's the hottest IT-related buzzword since the realization of the Internet. But there is a curious phenomenon associated with many people's view of the cloud—they somehow think that there is a unitary class of products and offerings that are all, more or less, comparable.

Of course, this view does not add much clarity to any consideration of the cloud; after all, what does the Apple iCloud storage have in common with *Salesforce.com*'s CRM solution? Before we can talk about Oracle and the cloud in any meaningful way, we should go over a few basic cloud definitions.

Common Characteristics

Virtually all cloud offerings have some common characteristics, which account for both their appeal and their challenges:

Subscription model
: The key driver for cloud computing is cost savings, and the key factor in that cost savings is the subscription model. Normally, you would use a large portion of your IT budget for initial capital expenses, including purchasing and installing hardware and a variety of system software, as well as the software you will be using for a particular project. With a subscription model, you simply pay as you go, with no (or relatively few) initial costs. Subscription models also include the ability to cancel at any time, although over time you may end up spending more overall when you *rent* rather than *buy* your IT resources.

Rapid time-to-value
: This somewhat vague phrase is meant to encompass a wide range of features that, taken together, mean that you can get to your desired end result of a cloud-based solution faster. This characteristic includes rapid, self-service provisioning, highly efficient configuration and deployment capabilities, and a variety of ways of increasing your operational productivity. All in all, it means the time from starting on a project to realizing business value from that project is significantly reduced, increasing the value of the cloud for you and your customers or business users. As the initial capital expenditure disappears in the cloud, so too does the initial setup delays to prepare an appropriate environment.

Universal access
: Although universal access has been around, in various forms, for awhile, the cloud essentially demands that you use Internet protocols to access the computing resources of the cloud. The use of the Internet as the communications level of the cloud has implications for the Oracle database, as you will see later in this chapter, but this type of access also has the important effect of making the location of cloud resources transparent. You should not really care where these resources are located, which means cloud vendors can use less costly locations for their infrastructure. There are significant and real factors that undercut this transparency, such as regulatory requirements and latency caused by distance, but the basic idea of transparency is central to a cloud solution. In the current environment, universal access also means access to the same systems from a variety of devices, from personal computers to tablets to mobile devices. Once this type of interface transparency is built into your cloud systems, you have guarded against obsolescence caused by any new platforms and user interfaces in the future.

Elasticity
: One of the key factors for the cloud is the ability to scale usage of resources up and down. The subscription nature of the cloud means that you will only pay for those

resources you use, so this elasticity is reflected in more efficient use of resources and the resulting lower cost. For databases, the focus of this book, elasticity mainly has to do with easy scaling up, since the amount of data typically only increases over time. However, even a database may use differing amounts of CPU and memory resources, so some cloud solutions offer differential rates based on fluctuations in these demands.

These four characteristics are common across virtually all cloud solutions. But all cloud solutions are not the same, as the following section details.

Cloud Levels

There are different levels of cloud products based on levels in the standard IT stack. These levels are, in ascending order in the stack:

Infrastructure-as-a-Service (IaaS)
: Provides hardware and system-level software at a basic level. The leading IaaS product is Amazon's Elastic Compute Cloud (EC2).

Database-as-a-Service (DBaaS)
: Provides a dedicated database in the cloud. The Oracle Database is available as a DBaaS from Amazon as one of the Relational Data Services (RDS) and from Oracle's Cloud Managed Services, among other vendors at the time of this writing.

Platform-as-a-Service (PaaS)
: Provides a deployment and, optionally, a development environment in the cloud. The Oracle Database Cloud Service provides an Oracle database as a PaaS product, and the Oracle Java Cloud Service provides a Java deployment platform. The *http://force.com* platform from *http://salesforce.com* is another example of a PaaS product.

Software-as-a-Service (SaaS)
: Provides a complete application system in the cloud. Oracle has a wide range of SaaS offerings, as do many other vendors.

All of these levels share two common, and crucially defining, characteristics:

The service level of an offering determines the interface to the service
: Each level utilizes the interface normally used by the software service level. For example, you would connect to an Oracle Database instance using SQL*Net. You would interact with the functionality of the Oracle Database from a development environment using SQL or PL/SQL. In a similar way, you use SQL*Net to access an Oracle database in a DBaaS offering. You would use SQL and PL/SQL to access an Oracle database used in a PaaS offering.

The service level of an offering determines the access you get to the component software
: In a nutshell, everything below the service level is hidden in the cloud. The downside of this characteristic is that you cannot, for instance, modify the configuration of

your Oracle database in a PaaS offering; that software is in the cloud and inaccessible to you. The upside of this characteristic is that you do not have to manage any software in the cloud, which reduces the overhead of your development and deployment environment.

Based on these two characteristics, you can understand that comparing products from different levels of the stack is inappropriate. If you want administrative control over the underlying database, for any reason, you cannot use a PaaS solution—you must go with either a DBaaS product or use an IaaS product as the foundation and build a database on top of it.

Similarly, if you want to create new cloud-based solutions and are interested in doing this as productively as possible, you would select a PaaS product, rather than a DBaaS solution—assuming that you would not need configuration options that were not available in a PaaS product.

You would not compare a list of features to see whether you wanted to use an Oracle database running on EC2 or the Oracle Database Cloud Service. Rather, you would examine your requirements and goals for your cloud project, which would lead you to the proper service level as the essential starting point for an evaluation. The different service levels give you a range of options, based on the particular requirements of your situation.

Is the Cloud New?

There is certainly excitement around the cloud, but is the overall class of cloud solutions really anything new, or is the C-word just the latest marketing phrase meant to revitalize existing products with just a name change?

The answer to this question is both yes and no, and the specifics have to do with which portion of cloud solutions you focus on. On one hand, the lower end cloud levels, such as IaaS and DBaaS, are similar to hosting, which has been around for a long time. Replace physical machines with virtual machines and these two categories are pretty much like that old war horse, and will provide the same benefits. At the end of the day, someone else is running your software and hardware. They had to pay for it, as well as pay for people to manage it, the same as you would, and they need to make their own margins. In addition, the lower down you go in the software stack, the more of the installation, maintenance, and license costs you are responsible for, so the lower the benefits to you. Of course, these options still have the advantage of no additional capital outlay and rapid provisioning, but these could also be realized with efficient hosting companies.

Multitenancy

Once you start dealing with PaaS and SaaS solutions, an additional technical factor comes into play, which goes by the name of *multitenancy*. As the name implies, multitenancy supports multiple tenants sharing a pool of resources. That simple definition

could apply to virtually any solution that runs on a general purpose computer, which shares resources among clients, or a server sharing resources between multiple virtual machines. But the key differentiator of multitenancy is a smaller granularity for sharing, which results in more efficient use of resources.

There are a couple of additional implications of multitenancy. One factor is that a multitenant architecture really doesn't matter—the underlying computer resources are hidden in the cloud, so what does a tenant care about whether a solution is multitenant or not? It's a good point, but the overall efficiency of a cloud offering has an impact on how the offering can be priced and how well it scales.

The other implication is that multitenancy, whether done with schemas (for) or some other architecture, is different from traditional architectures. This difference has an impact on whether application systems that have been designed for different architectures may not work optimally, or at all, in these new environments.

Stateless

Another key technical factor comes with the introduction of the Internet, which runs on the HTTP protocol. HTTP is a stateless protocol, which means that each communication using this protocol is a separate transaction. Since one of the characteristics of the cloud is the use of the Internet, cloud solutions are also stateless by nature.

This stateless nature may not make a big difference, as transactions are frequently restricted to a single communication as a good design practice. But sometimes this limitation can have a significant impact. Consider the case of an application that returns a page of results from a multipage result set. The page looks fine, with appropriate Next and Previous buttons to scroll through results. But be aware that each request for another page runs the underlying query again, in a different transaction. Any changes that have taken place since the previous run of the query will be in the new result set. So if the first page contained 10 rows, and one of those rows was deleted between the fetch of the first page and the fetch of the second, the second page would not start with the 11th row from the previous result set, but with what had been the 12th, since one row is now gone.

An Oracle Database, whether in the cloud or not, has a way to overcome this stateless limitation, at least for queries.When you ask for the first page, you can retrieve the System Change Number for the transaction, and then use this for a Flashback Query for subsequent requests to emulate full stateful data integrity over the stateless HTTP protocol.

Depending on the nature of the application, this type of potential integrity hole may not matter. However, the possibility of this type of issue has an impact on the potential use cases for the cloud, as discussed in the next section.

Use Cases for Cloud Computing

The cloud seems like a way to save money, at least in the short term, and get much faster results. So run, run, run to the cloud. Take your entire data center and move it to the cloud. Well, maybe, not so fast.

There are some inhibitors to keep you from moving entire systems to the cloud. The first is that moving to the cloud, like moving to any new platform, is a migration effort. As such, you will have to ensure that your system works the same on the new platform as on the old platform. This comes with a cost—at minimum, a significant round of testing, frequently extended with the need to correct problems that you find.

Application systems, which have not been designed to benefit (or even operate) in a multitenant architecture, may not transfer so well, and the stateless nature of cloud interactions may also interfere with the proper operation of systems.

Moving your data to the cloud may cause fewer problems, since the data will end up in an Oracle database. There are still a number of potential issues here, such as the time required to move data to the cloud. With even a big and dedicated network pipe, it could take more than a day to move 1 TB of data to the cloud, and you would still have to test to ensure the success of the transfer. There may be regulatory requirements or organizational issues that would prevent this type of data movement from your data center. There are also potential runtime issues, since your network connections to Oracle DBaaS would go over the Internet, which has both high latency and unpredictable routing. Together, these characteristics could add up to poor performance. The impact of these issues can be reduced by using things like dedicated leased lines and virtual private networks, but even these solutions may not match the performance you are accustomed to, and the additional cost and overhead of these components reduce the economic and productivity benefits that led you to the cloud in the first place.

And both application deployment environments and an Oracle database running in a shared cloud will inevitably be subject to some lockdowns for security and prevention of any one tenant from interfering with the overall performance of the system. These limitations could force you to make modifications to existing systems, adding to your migration overhead.

All in all, migration from an on-premise solution to the cloud frequently ends up being less attractive than initially thought, based on the costs of migration.

The best way to avoid migration costs is to simply avoid migration. The cloud is an ideal environment for creating new systems, where the rapid time-to-value of provisioning and environment and developing applications has immediate benefits. And, although

the limitations of the cloud may seem constraining when trying to migrate a system that was not designed for that platform, most robust cloud offerings allow you to create rich and even enterprise-strength applications in the cloud.

We believe the optimal cloud strategy is to start to use cloud computing to create new systems, rather than expending resources to move existing systems to the cloud. Over time, your overall IT workload will start to shift towards the cloud, coincident with your growing in-house expertise in using the cloud.

Oracle Database in the Cloud

With the previous definitions and implications understood, we can move on to the specific case of the Oracle database in the cloud. In fact, the Oracle Database is accessible in two different cloud levels: DBaaS and PaaS.

This duality typically takes people by surprise. After all, doesn't the Oracle database in the cloud belong in the DBaaS level, since it is the world's leading enterprise database?

Oracle as a DBaaS

Oracle offers two DBaaS solutions for the Oracle Database. Oracle Cloud Managed Services was previously known as Oracle OnDemand. This offering gives customers their own instance of an Oracle database, running on Oracle hardware in Oracle facilities. The database is accessible through SQL*Net, just like a standard Oracle database instance. You have flexibility in how you configure and manage this database, since you own the instance. As with other cloud products, you pay for the service on a monthly basis.

As of the time of this writing, Oracle is in the process of rolling out another variation of DBaaS where the actual hardware sits in your data center. This service will run on Exadata and will be fully managed by Oracle, with most management operations being performed remotely by Oracle.

Other vendors also offer the Oracle Database as a DBaaS, most notably Amazon. Amazon has an RDS, or Relational Data Service, built on the Oracle Database. This RDS offering comes with the Oracle database included, so you don't have to bring your own license.

Oracle as a PaaS

Oracle also has a significant offering in the PaaS space, Oracle database Cloud Service. You may find this a bit confusing, but remember that the Oracle database is the foundation of an enormous number of applications—the Oracle Database already acts as a deployment platform, and has long included development tools that have been used to

build hundreds of thousands of applications, from quick and easy departmental solutions to rich, enterprise-wide systems.

The Oracle Database Cloud Service is described in detail in the following section. Before getting to the specifics, we should understand the difference between two types of clouds.

Consumer and Provider

There are two common words used in cloud discussions that you have not seen in this chapter yet: public and private. We don't really like these words, because they are overloaded with different and somewhat confusing meanings.

Many people use the term *public* to indicate a cloud that runs offsite, and the term *private* to indicate a cloud that runs in your data center. But what about a cloud that runs in your data center but is owned, managed, and controlled by a vendor? Or a cloud that runs offsite but is exclusively for your use? And isn't one of the core ideas of the cloud that the location of the supporting infrastructure should be transparent?

There is a difference between different types of cloud usage that we like to call consumer and provider. A consumer cloud offering is one where you simply use the cloud services. As described earlier, you do not have access to any of the software or the hardware inside the cloud—you simply take the product as offered. A company can have different varieties of consumer cloud offerings at the same service level, but each of them is fixed in terms of the underlying components.

You can also use those underlying components to build your own cloud offering—you would be acting as a cloud provider. You get to choose the options and configurations of your consumer cloud offering, and you get the privilege of designing and implementing provisioning, maintenance, and upgrade operations. Your consumers could be entirely internal customers to your organization, but they would have the same ease of use and lack of flexibility as other users of consumer cloud products.

The remainder of this chapter will discuss the main Oracle consumer cloud product, the Oracle Database Cloud Service, as well as the ability to use the Oracle Database as the foundation for cloud providers.

Oracle Database Cloud Service

The Oracle Database Cloud Service is a PaaS offering. The Database Cloud includes a complete development and deployment environment, based on the longstanding product Application Express.

The Database Cloud is fully managed and available within minutes. You do not have to perform installation or ongoing maintenance for the platform. A Database Cloud subscription is all inclusive, with no additional costs for support or any other feature.

The pricing for a Database Cloud Service is also straightforward. There is essentially one metric—storage—and three options for that metric.

At this time, the pricing for the Database Cloud Service is:

- A 5 GB Oracle Database Cloud Service for $175 a month.
- A 20 GB Oracle Database Cloud Service for $900 a month.
- A 50 GB Oracle Database Cloud Service for $2,000 a month.

In addition, you can get a 30-day free trial of the Database Cloud Service with a limit of 1 GB of storage.

Why Mention Pricing?

This book is focused on the concepts behind Oracle technology, so why mention pricing here, for the first time in this book? The only reason is to provide an illustration of the general level of cloud pricing, which most readers will immediately see is significantly different from Oracle's license policy. The pricing cited is current, as of this writing, for purchasers in the United States, although similar pricing is in place throughout the world.

Subscriptions can be either month-by-month or for a term of one or more years, and discounts are available for term licenses.

There is another metric associated with a Database Cloud Service—data transfer—which refers to the data requested by Database Cloud application users. The monthly transfer amount is six times the storage allocation. At the time of this writing, Oracle is tracking the data transfer amounts but is not charging for any overages. Oracle's experience with the public-facing *http://apex.oracle.com* site indicates that the data transfer allowance should be enough for legitimate uses of the Database Cloud Service.

You may wonder why there are no metrics associated with the type of resources that typically have an impact on performance, such as CPU cycles or memory. Remember that the Oracle Database Cloud Service is a PaaS, so you, as a tenant, would not get access to resources used by underlying software. Further, remember that one of the main functions of the Oracle Database is to share resources between many, many users. Much of what has been discussed in this book previously centered on how Oracle efficiently shares resources between users, from the SGA to pushing CPU processing to storage in Exadata machines. Since tenants in the Oracle Database Cloud are isolated on the basis of schemas, the Oracle database uses all the standard resource sharing between tenants that is used for normal database users, who are also associated with schemas. And the Oracle database is very good at this type of sharing.

The one factor that could impact performance in a shared environment like the Database Cloud is the possibility that one tenant could use too many resources, denying the resources to others. Some of the lockdown of the Database Cloud, discussed in more detail below in terms of security, is concerned with preventing this possibility. Any action that could soak up too many resources is either disallowed or limited, such as the limit on the number of database jobs each tenant can spawn. Additionally, the Database Cloud uses a series of Database Resource Manager consumer groups to reduce the priority of any user who exceeds certain thresholds of CPU usage. This approach means that virtually all requests are complete, but that requests with excessive requirements sacrifice some of their resource availability for the good of the overall Database Cloud community.

The Database Cloud Service is built on the foundation of Oracle Application Express, commonly referred to as APEX. The Database Cloud is a relatively new offering, but APEX has been a part of the Oracle Database technology stack for a while.

History of Application Express

The product known as Application Express began in the late 90s in an Oracle office in the Washington, DC area. Two sales consultants were interested in the then new environment of the Internet, and HTML, which was the language used to construct pages retrieved over the Internet. The Oracle database had a feature that allowed generation of HTML from PL/SQL, so the two intrepid consultants got to work creating a development and deployment environment that could be used to create HTML pages and serve them up to users from an Oracle database. They also created a listener that would handle URL requests for these pages.

By 2001, a version of their work, then known as WebDB, was in use inside Oracle for a variety of applications. The WebDB product underwent a transition, and another version of the same basic tool was created with the name HTML-DB. This product was more widely adopted, both inside Oracle and by customers, who could use it with the Oracle Database. The product was renamed Application Express in 2004.

In 2004, Oracle also started the *http://apex.oracle.com* website. This site gave developers a sandbox environment to create APEX applications.The site provided a fairly limited amount of data storage (50 MB), was specifically restricted to development, rather than production work, and was offered free of charge, as it still is. The *http://apex.oracle.com* site was essentially a PaaS platform before anyone had even thought of the label.

APEX itself has seen fairly wide adoption in the Oracle community. The *http://apex.oracle.com* site currently has more than 14,000 individual workspaces. APEX has been included as a free option for all editions of the Oracle Database since the Database 10*g* Release 2, so estimating the worldwide usage of APEX is a bit difficult. However, a "phonehome" check for updates was implemented in 2012, and in the first half of that

year, more than 160,000 unique development sites called in. Considering the fact that these are development sites, rather than developers, and you can see that a pretty healthy APEX community is out there.

Architecture

As mentioned previously, the architecture of any cloud offering should be irrelevant to users—that architecture is hidden in the cloud. But some facets of the architecture of the Oracle Database Cloud are worth understanding, as they have implications for the use of the service.

Database Cloud architecture

The Oracle Database Cloud Service has all the features normally expected in the user interface to a cloud computing product. You can get a service in a matter of minutes through a self-serve process that starts at the general portal for all Oracle Cloud offerings, *http://cloud.oracle.com*.

The Oracle Database Cloud runs on a full Exa-stack, with the database running on Exadata and the APEX Listener running on a WebLogic Server on an Exalogic machine. Remember, the Database Cloud is a PaaS product, so the fact that your service uses both an Oracle database and a WebLogic Server is not some special dispensation—you need both to support the platform.

At the time of this writing, the Database Cloud uses Oracle Database 11*g* Release 2 Enterprise Edition, the most current database release. Oracle has stated that they will always support at least two major versions running on the Oracle database cloud, so the appearance of Database 12*c* means tenants have the option to upgrade to this version or not, at least until another major version is released. The only option available for customers to use in the Database Cloud at this time is partitioning. The Database Cloud also uses Transparent Data Encryption, which is a part of the Advanced Security Option.

Each individual Database Service is completely isolated from all other Database Services. The Database Cloud implements multitenancy on the basis of a schema—each tenant gets one and only one schema for each service. Because of this, the Database Cloud does not support a number of features such as database links and synonyms. For more information on restrictions on the Database Cloud, please see the section "Security and architecture" on page 362.

> Please note that this single schema restriction is enforced on a Database Cloud Service, but not on the equivalent workspace for a version of Application Express running on your own instance. APEX workspaces can be configured to access multiple schemas.

Access architecture

The Database Cloud Service is a PaaS product. As such, you have access to the capabilities of the Oracle database through SQL and PL/SQL, which are executed by applications running in the Oracle Cloud. At the time of this writing, there are only two platforms supported in the Oracle Cloud: Application Express in the Database Cloud and Java in the Java Cloud. You can only run applications written for these two environments against your Oracle Database Cloud Service and none other.

You can access your Database Cloud Service from outside the Oracle Cloud, but only through RESTful Web Services. These Web Services are described in more detail later in this chapter. Remember, you cannot access a PaaS with a database network protocol like SQL*Net. So you cannot move your Oracle database to an Oracle database Cloud Service and then simply create a new entry in your *TNSNAMES* file to point to the cloud.

Security and architecture

The Oracle Database Cloud uses a multitenant architecture based on schema isolation. Because of this, some syntax, built-in packages, and capabilities are not allowed in the Database Cloud. For instance, you do not have the ability to read or write to the underlying filesystem, due to the potential for hacker mischief.

In addition, some capabilities could be used to swamp the underlying servers in a denial of service attack, so other features have had to be limited or prevented. The Database Cloud development team is working to minimize the impact that these security-based restrictions have, but the priority is protection of tenants, their data, and their environment. A white paper that lists the specific areas locked down is available at *http://cloud.oracle.com* under the Resources menu.

As mentioned above, the Database Cloud Service does use Transparent Data Encryption to protect all data at rest, as well as requiring HTTPS for communication from clients. RESTful Web Services have their own set of security capabilities, which are also described in a white paper on RESTful Web Services at the main Oracle Cloud site.

Each individual Database Cloud Service is unique and fully isolated, but there is an organizational hierarchy with the Database Cloud as shown in Figure 15-1. When you initially request a trial or purchase a Service, you act as the account owner. All account owners must have a valid login for *http://www.oracle.com*. When you initially create an account, the cloud creates an administrative user for the account in the Cloud Identity Manager with the same username and password as your *http://oracle.com* identity.

Each account can have one or more Identity Domains associated with it. An Identity Domain is a group of users. By default, the administrator of an Identity Domain is the administrator of the account that owns the Identity Domain. You can give administration privileges for an Identity Domain to one or more users defined in the Identity Domain.

Each Identity Domain can have one or more Database Cloud Services associated with it. When you create a Database Cloud Service, you assign a Database Cloud Service administrator to the service. Once again, by default, the Identity Domain administrator is given administrator privileges for any Database Cloud Service associated with it.

All users in an Identity Domain potentially have access to all Database Cloud Services associated with that Identity Domain, although you have to give specific privileges for a Database Cloud Service to users.

Finally, a Database Cloud Service can have three levels of user privileges: an administrator, who has full control over the Database Service; a database developer, who can create applications or Web Services in the Database Cloud Service; or a user, who can simply run applications in the service. You can also designate applications to allow all users to access them by making the applications public.

All administrators and database developers must be defined in the Cloud Identity Manager. Application users can be defined in the Cloud Identity Manager or within the Application Express administration area.

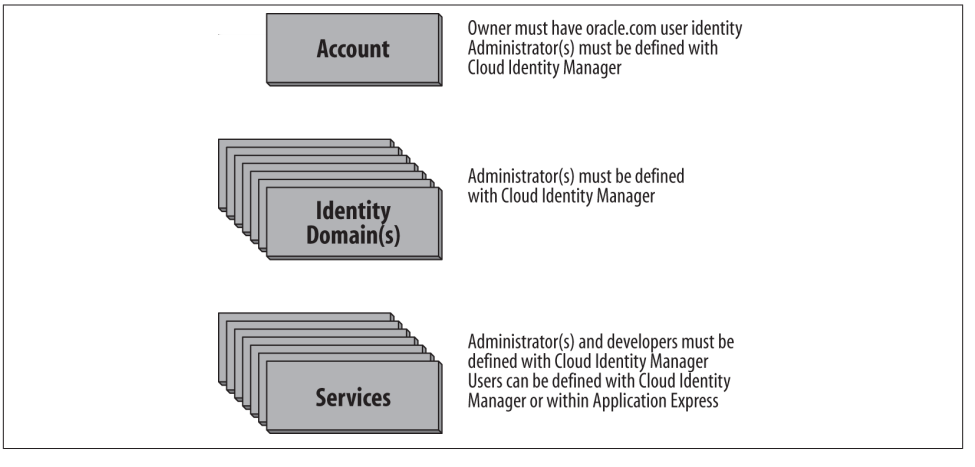


Figure 15-1. Domain levels and security for the Oracle Database Cloud

APEX architecture

The core of the Database Cloud is Oracle Application Express, or APEX. APEX, from its inception, has been a true multitenant environment that uses the resource-sharing capabilities of the Oracle database for optimal efficiency.

APEX runs entirely within the Oracle database, although APEX requires some type of listener to receive incoming HTTP requests and respond to them.

There are three options for listeners for APEX: an embedded PL/SQL listener, a mod that can be used with Apache, and the APEX Listener that is used for the Oracle Database Cloud Service. Some capabilities of the overall environment discussed here, such as RESTful Web Services, are dependent on the use of the APEX Listener. The Database Cloud Service includes the APEX Listener.

APEX is a declarative development environment. When you create applications, pages, or components that are shared across pages or Web Services, APEX uses wizards to collect metadata. When a particular page is requested by a user, a PL/SQL procedure retrieves the metadata and constructs the HTML page based on that metadata.

This architecture has two important implications for APEX applications. The first is that applications are just a set of metadata, which means you can move them from one Oracle instance to another with a simple SQL script. As long as you have the metadata repository tables and the PL/SQL procedures installed, the application will run on the target instance.

The second implication is more profound. Since pages are built dynamically from metadata, there is absolutely no deployment process for APEX applications. This architecture, coupled with the high productivity of a declarative development process, leads to the capability to do *iterative development*. You are able to not only create applications rapidly, but you can change them in real time while working with users. The interaction with users and the high productivity are well suited to the world of cloud computing.

Development with the Database Cloud Service

As mentioned above, Application Express has been around since 2004, with a fairly large and very involved user community. Over the years, Application Express has become a very rich and robust development environment.

Since APEX is based in the Oracle Database, you can use the full power of SQL and PL/SQL to define data interactions and extend application logic, respectively. Although you do not have to know either of these interfaces to create and deploy APEX applications, the ability to use them means that you can build virtually any type of application with APEX. Since APEX applications are made up of HTML pages, you can also use features from HTML, including formatting and JavaScript, in your APEX applications, although you do not need to know HTML to create applications either.

Each APEX page includes metadata to control the appearance of the page and any special processing for the page, and typically uses a set of shared components across pages.

APEX application development

APEX applications are a set of pages, with navigation capabilities bringing them together into an integrated unit. There is a broad set of wizards you can use to create APEX pages. Figure 15-2 shows the top level of page wizards in the APEX development environment.

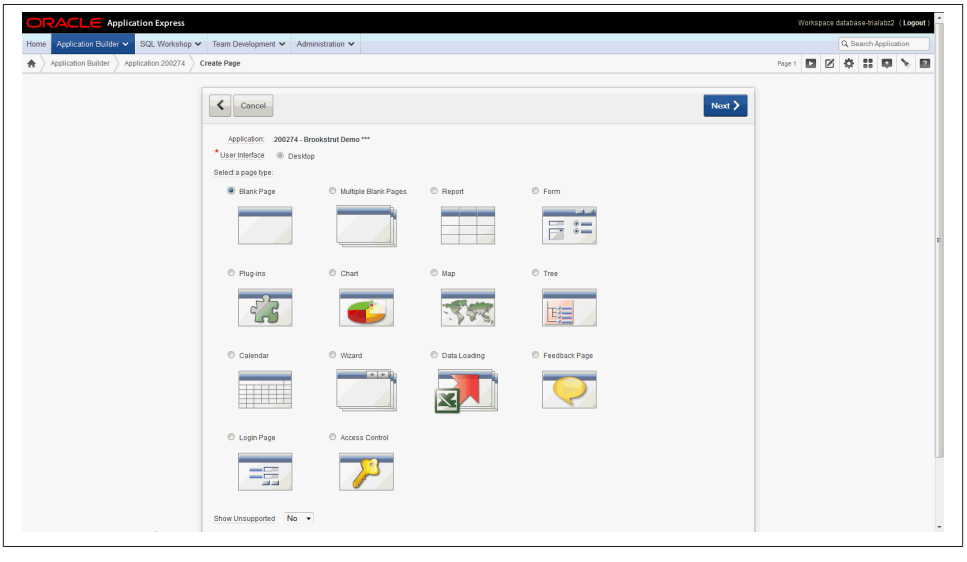


Figure 15-2. Oracle Application Express page types

A page can contain more than one of the types shown in Figure 15-2, with different regions in the page containing different components. Also, each of the categories of pages shown in Figure 15-2 can have multiple types. For instance, you can create a form based on SQL, PL/SQL, or a web service, or a master detail form, or a report with an associated form, or a form with multiple editable rows, to name just some of the form options.

One type of component merits specific mention: the interactive report. An interactive report includes a wide range of customization capabilities, including the ability to define filters, sort data, create new columns from data in the report, format the report with control breaks and groupings, highlight data based on logical conditions, and create charts based on the data in the report. You can even look at data in the past with Flashback, download data in spreadsheet format, or allow subscriptions to a report so that the report will be sent to subscribers according to a schedule. Each individual user can customize their own versions of reports and save these versions for exclusive access or public use. As a developer, you can disable any of these options, depending on your requirements. Interactive reports enable users to shape reports to get answers on their own, which can remove a significant development overhead.

All APEX applications use themes and templates to create a uniform look and feel to pages and their components. APEX comes with a set of themes, as well as the ability to define your own theme. You can use this concept to create applications with mobile interfaces by either using a theme based on responsive design principles, which allow you to dynamically resize and reposition components based on the dimensions of the calling device, or by creating a mobile interface, which uses specific mobile-type interfaces, such as list views. An application can have a desktop interface and a mobile interface, with automatic selection of the interface based on the identity of the calling device.

Individual pages are the building blocks of APEX applications, but an APEX application also uses components that are shared across many pages in the applications. These components include:

- Navigation components, such as tabs, lists of links, and breadcrumbs
- Themes and templates, which control the look and feel of the overall application
- Application-level computations, processes, variables, and options
- Various authorization and authentication options

APEX applications are composed of HTML pages, which when submitted interact with the Oracle Database hosting the applications. APEX also gives you the ability to declaratively implement dynamic actions, which can operate in the browser environment through generated JavaScript.

APEX also includes team development capabilities, which help team members to communicate about projects, issues, and bugs. One additional and useful feature of APEX is the ability to assign a page or other object to a particular build option. You can then specify what build options are active at any time. This capability lets developers work on new options, but leave them out of export scripts created to move an application to a production environment.

SQL Workshop

The previous portions of this section described, at a very high level, the application development capabilities of Application Express and the Oracle Database Cloud. The Database Cloud and APEX also provide an area called SQL Workshop, designed for working with the data structures and data in the underlying Oracle Database.

SQL Workshop includes the following main areas:

Object Browser
: Lets you see the structure and contents of your data objects. The Object Browser also includes the ability to create and modify objects.

SQL Commands
: A text box where you can enter any SQL statement and show results. This area also can show you EXPLAIN PLAN or DESCRIBE output and includes a list of recently executed statements for rapid re-execution through links.

SQL Scripts
: Allows you to upload scripts for storage in your Database Cloud Service and execute them. You can use this capability for loading data into your Database Cloud Service, although SQL Developer, described below, is more appropriate for larger amounts of data.

Utilities
: Includes a number of useful tools, including: a graphical query builder, which is also accessible from different wizards in the application development area of APEX; a data workshop, for importing and exporting data; and tools to help you with common tasks, such as comparing schemas, creating PL/SQL packages for access to tables, and getting reports on objects in your database. You can also set user interface defaults on tables and columns that are used by the APEX application development wizards when you create new application components.

The SQL Workshop area also includes an area for working with RESTful Web Services, which includes a wizard that makes it easy to define and use RESTful Web Services for your Database Cloud Service or Oracle database, through the APEX Listener. This functionality is described below.

Packaged applications

All versions of APEX, including the Database Cloud, come with a set of packaged applications. These applications are production-ready and can be installed with a handful of clicks, emerging ready to run in minutes. Although packaged applications are locked by default, preventing access to the underlying application, you can unlock them to extend or modify the application, or simply to see how the packaged application operates.

APEX also includes a number of sample applications that provide examples of implementing various techniques in your own applications.

RESTful Web Services

Earlier, we mentioned that you can access the Database Cloud from within the cloud using APEX applications or Java applications running in the Java Cloud. You can also use RESTful Web Services to access your Database Cloud Service from outside the Oracle Cloud.

The RESTful Web Service interface is a specification for a lightweight web service that has become very popular on the Internet. A RESTful Web Service uses a URI to access a specific web service. In this way, using RESTful Web Services is different from using

an API like SQL*Net, which has the ability to simply pass some SQL syntax to an Oracle instance. With RESTful Web Services, a URI calls a specific SQL statement or PL/SQL block in your Database Cloud Service.

APEX includes a wizard to define RESTful Web Services. You can specify that RESTful Web Services return data from queries in either JSON (a common web format), or as a *.csv* file. The APEX Listener will properly marshal the data being returned. You can also choose to have data sent back without intervention, which is best when you are returning media files.

There are a number of ways you can add security to your RESTful Web Service calls, including the use of OAUTH2, a popular Internet authentication protocol, as well as using logic within a RESTful Web Service call to limit access to data.

Portability with the Database Cloud Service

You can create and deploy applications in your Database Cloud Service, and you can use the underlying Oracle database to store data and implement logic with PL/SQL. Since the technology for this functionality is contained within the Oracle Database, you can easily move your data and the metadata for applications and RESTful Web Service definitions from the Database Cloud to any Oracle instance. The Database Cloud management console gives you the ability to export all of your data and definitions with a single click. The export file is placed on a secure FTP server, where you can retrieve it and simply load it into another Oracle Database as you would any other dump file.

You can export application metadata from within the APEX environment, or use a packaged application to archive your Database Cloud applications into SQL scripts. Similarly, you can export RESTful Web Service definitions into an SQL script. This architecture makes it easy to move from the Database Cloud to any other Oracle database instance.

However, moving an application from an Oracle instance outside the cloud into your Database Cloud Service may not be as clean. In order to protect the security and integrity of every tenant of the Oracle Cloud, Oracle has had to lock down some features and functionality. For instance, since the Database Cloud uses schemas for isolation, you do not have access to instance-level configuration options, or even internal tables and views that grant visibility over multiple schemas. In addition, some features, such as the ability to write directly to the filesystem of the server, have been blocked due to the potential for hacker mischief.

A proposed idea called "cloud bursting" has little chance of working properly in the real world. This concept calls for applications to automagically spill over into the cloud once they require resources beyond those available on-premise. It's a nice idea, but the reality of having to coordinate transactional changes between completely separate data stores makes this dream extraordinarily difficult to implement in the real world. If you have designed your application from the ground up to work in this manner, you have a

chance, but even this requires significant design and implementation considerations. With a PaaS solution, such as the Oracle Database Cloud, your optimal path is to create new systems in your Database Cloud Service, avoiding the overhead and complications that come with any migration effort.

SQL Developer and the Database Cloud

SQL Developer is a very popular tool used to view and manage Oracle database instances. SQL Developer, which is part of the same development organization as APEX, has been modified to allow for connections to an Oracle Database Cloud Service. At the time of this writing, this cloud connection has a more limited scope of functionality than a connection to a direct-connect Oracle database. The connection to your Database Cloud Service allows you to see the tables and other structures, but without the advanced capabilities of a standard connection.

You can use SQL Developer to load data to your Database Cloud Service, especially larger amounts of data. You start the process by creating a cart and then dragging the objects you wish to move from a connected database to the cart. Once you have loaded the cart, you simply click on the Deploy Cloud button to start the process of moving all the data structures and/or data to your Cloud Service.

The data is exported from the connected database and then compressed and moved to an SFTP server associated with your Database Cloud Service. Once on the SFTP server, a periodic process picks up the file, scans it for viruses, decompresses the file, and loads it into your Database Cloud Service. You can move up to a million rows a minute with this process, which makes SQL Developer the appropriate choice for moving larger amounts of data to your Database Cloud Service.

Implementing Provider Clouds

As described at the start of this chapter, the Database Cloud Service is a Platform-as-a-Service consumer cloud, where you can quickly provision and use the service for creating and deploying applications. Oracle's enterprise software can also be used to build your own provider clouds. As with any provider cloud, you can specify what exactly you want in your cloud. In that sense, the rest of this book applies to what you can put in your own cloud.

There are a number of features in the Oracle Database that lend themselves to the creation of provider clouds. Of course, all the capabilities discussed in this book contribute to the overall richness of the Oracle Database and any cloud built using this database. Oracle Database 12*c* includes the Oracle Multitenant Option and pluggable databases, which can help in creating provider clouds in a number of ways. First of all, the multitenant architecture is a way to implement multitenancy within a single Oracle instance, which can be mapped to individual cloud consumer tenants. Pluggable databases are

easy to clone, which can make provisioning tenants faster and easier. And pluggable databases can be used very effectively in a popular use case, where a test or development instance is used in a cloud and then moved to a different platform for production use.

There are some other features that lend themselves specifically to building your own Database Cloud. Enterprise Manager has a number of cloud-focused features, including self-service provisioning of seed databases, chargeback monitoring and tracking to allow for tenant billing, and Schema-as-a-Service provisioning, similar to the Oracle Database Cloud. However, you can enable access to more than one schema for a user, more like on-premise APEX. This service can also utilize Data Vault for security and Database Resource Manager for resource limiting, as well as assigning quotas, retirement policies, and associating services with chargeback plans.

Enterprise Manager also provides a "snap clone" feature, which uses database snapshots to create database clones. This process results in much faster provisioning for seed databases.

As with the Database Cloud, you would probably want to use Database Resource Manager to prevent any one tenant from using too many resources and impacting other users. Keep in mind that Database Resource Manager has a lot of flexibility as a standalone product, as was described in Chapter 7. You can not only limit CPU resources for users, but many other resource dimensions, including degree of parallelism and I/O bandwidth on Exadata machines.

The Oracle Database Cloud is designed to support a wide variety of general purpose cloud tenants, and so can only make fairly generic use of Database Resource Manager. For your own provider cloud, which you design for what could be a well-defined set of user requirements, you could be more exact in your use of this product.

You may choose to use RAC for your own provider cloud, either for fast failover, as in the Database Cloud, or for the other scalability-related features of RAC. Similarly, you could use DataGuard to provide high availability and failover for your cloud, if appropriate. The latest version of DataGuard, with its ability to use a failover machine located in a distant location with far synch, could also be used in your cloud.

Of course, you can also use Application Express as part of your provider cloud to include all the development capabilities of the Oracle Database Cloud Service in your own cloud. Depending on your own scenario, you could offer APEX without some of the limitations that Oracle must impose for a consumer cloud offering designed to be used by thousands of unrelated clients.

In closing, remember that you will have to take care of all the operational aspects of your provider cloud, which will involve creation of standardized operations to service all your cloud clients. The productivity and robustness of the Oracle database, probably the most sophisticated database in the world, is there for you to use to offer the most appropriate cloud for your particular customer needs.

APPENDIX A

What's New in This Book for Oracle Database 12*c*

When we wrote the first edition of *Oracle Essentials* in 1999, our goal was to offer a new kind of book about Oracle, one that would clearly and concisely cover all of the essential features and concepts of the Oracle Database. In order to keep our focus on those essentials, we limited the scope of the book.

For instance, we decided not to cover SQL, or PL/SQL, in depth; these complex topics would have required a level of detail that would have run counter to the purpose of our book, and they are amply described in other books.

The latest release of Oracle, Oracle Database 12*c*, contains many new features. Most of these features build on the existing foundation of Oracle technology and enable new flexibility in deployment. We tried to add details about these features in the chapters in which their discussion seemed most appropriate, but there are (of course) some enhancements that are outside the scope of this book.

The following sections summarize the new features of Oracle Database 12*c* that are covered in this new edition, chapter by chapter. Although many of these features are mentioned in multiple chapters, they are listed here according to where the most relevant discussion occurs.

Chapter 1: Introducing Oracle

This introductory chapter was extensively updated to reflect the packaging changes in Oracle Database 12*c* and introduce many of the new features described in more detail in other chapters. Given Oracle's acquisition and development strategy, there were also many changes to software and technologies that surround the Oracle database and many of those components are described here.

Chapter 2: Oracle Architecture

This chapter describes concepts and structures at the core of the Oracle database. Features new or modified in Oracle Database 12*c* or changed and mentioned here include:

Multitenant Container Database (CDB)
: Operations and entities that apply to an Oracle instance are implemented here and are intended to operate across multiple separate Pluggable Databases (PDBs).

Pluggable Database (PDB)
: Pluggable databases are plugged into container databases (CDB) and contain the database schema. PDBs are useful for database consolidation, provisioning of new databases, copying existing databases, and upgrades or patching.

Database Resource Manager
: Now also manages resource allocation between PDBs.

Minimum Initialization Parameters
: Only CONTROL_FILES, DB_NAME, and MEMORY_TARGET need be specified. Others parameters are set to default values.

PGA_AGGREGATE_LIMIT
: A parameter used to place a limit on the total memory available to the PGA.

Chapter 3: Installing and Running Oracle

Although the standard installation and runtime operations of the Oracle database remain essentially the same, the installation process was improved such that it could take only 20 minutes on a desktop configuration. A few Oracle Database 12*c* enhancements covered in this chapter include:

Global Data Services
: Enables widely geographically dispersed databases to participate in a single global service.

Database Resident Connection Pooling
: Connection pooling implemented in the database server instead of a middle tier.

Flashback Query improvement
: Flashback query support extended to queries on temporal validity dimensions.

Chapter 4: Data Structures

This chapter covers the basic data structures and optimization technology in the Oracle database. New features include:

Maximum Column Size with VARCHAR2 or NVARCHAR2
: The maximum column size with VARCHAR2 or NVARCHAR2 can now be reset to 32K.

Temporal Validity
: Enables specification of a time period for when a row or rows of data in a table are valid.

RAW Datatype Column limit
: A RAW datatype column can now hold 32K.

Identity Datatype Support
: A new datatype intended to ease migration from IBM DB2 to Oracle user-defined data.

Invisible Columns
: Columns that are stored and maintained like regular columns but not accessible by users or considered during query optimization.

Interval Reference Partitioning
: A new Oracle Partitioning Option composite consisting of interval partitioning (automatic creation of range partitions) and reference partitioning (parent-child).

Statistics generation when needed during SQL statement compilation
: During compilation of SQL statements and creation of the execution plan, will automatically generate new statistics if they are missing or out of date.

Adaptive Plan Management
: Multiple plans may be specified for query optimization and then be selected from depending on user directives or runtime statistics.

Chapter 5: Managing Oracle

This chapter covers Oracle Enterprise Manager 12*c* and other manageability features new in Oracle Database 12*c* including:

Real-time ADDM
: Runs automatically every 3 seconds enabling DBAs to resolve deadlocks, hangs, shared pool connection issues, and similar situations with needing to restart Oracle.

SQL Plan Management Evolve Advisor
: Enables scheduling of testing of new plans added to the SQL plan baseline, compares new plans versus accepted plans for cost, and automatically selects new plans that have a much lower cost.

ASM improvements
: ASM disk scrubbing provides automatic repair of logical data corruptions. ASM disk resync enables multiple disks to be brought back online simultaneously.

Enterprise Manager Express
: A version of Enterprise Manager that doesn't require middleware.

Backup and Recovery enhancements
: Recovery of individual tables is supported as well as backup and recovery of multitenant container databases and point-in-time recovery of individual pluggable databases.

Automatic Data Optimization (ADO) and Heat Maps
: ADO provides the ability to automate compression and movement of data among different tiers of Database storage and leverages the new Heat Maps.

Chapter 6: Oracle Security, Auditing, and Compliance

This chapter covers Oracle security and related features. Some significant new features available with Oracle Database 12*c* include:

Privilege Analysis
: Identifies what users have broad privileges and don't need them.

Real Application Security
: Users are identified outside of database user context and assigned appropriate security privileges.

Data Redaction
: Ability to obscure data returned in queries.

Auditing flexibility
: Auditing of database activities can be policy based. Records can be written immediately or every few seconds to minimize resource consumption.

Oracle Database Vault enhancements
: Mandatory realms added to block users who were granted access.

Oracle Audit Vault enhancements
: The Database Firewall product is included with Audit Vault.

Chapter 7: Oracle Performance

In addition to Oracle database management improvements previously noted, performance tuning improvements in Enterprise Manager 12*c* including additional Active Session History (ASH) and ADDM reports. This is also the first edition of this book covering the Exadata Storage Servers describing optimizations provided there. (Optimizations for data warehousing are further described in Chapter 10.)

Chapter 8: Oracle Multiuser Concurrency

The ability to handle very large groups of users without excessive contention, while protecting data integrity, has long been one of the best features of the Oracle database and a core part of the Oracle database for over 20 years. This chapter also covers workspaces—the query optimizer was further improved in Oracle Database 12*c* in support of them.

Chapter 9: Oracle and Transaction Processing

Oracle has been one of the leading databases for OLTP for many years. New features in Oracle Database 12*c* include:

Transaction Guard
: Provides an API enabling developers to build code that determines a transaction's state.

Application Continuity
: Enables creation of database requests to take action using information provided by Transaction Guard.

Chapter 10: Oracle Data Warehousing and Business Intelligence

In addition to covering the Oracle database for data warehousing, this chapter describes Oracle's current suites of business intelligence tools and business intelligence applications. New in this edition is a description of the Advanced Analytics Option, the role of Big Data and Hadoop, and descriptions of how engineered systems are used for query optimization and analyses. For example, how Hybrid Columnar Compression and flash in Exadata are used to improve query performance is described. New features in Oracle Database 12*c* include:

Data Mining enhancements in the Advanced Analytics Option
: A new probabilistic clustering algorithm, Expectation Maximization, was added to solve problems such as finding the most representative customer in the largest cluster. Other improvements to Data Mining features include Generalized Linear Model feature selection and creation, and simplified model building, deployment, and testing for unstructured text data.

SQL Pattern Matching
: SQL capability added for matching patterns in multiple rows of data tied to a data item of interest.

Chapter 11: Oracle and High Availability

This chapter describes the Oracle characteristics that keep your database up and highly available. New features include:

Flex ASM
: Enables ASM servers to be run on a separate physical server from Oracle Database servers.

Data Guard enhancements
: Data Guard can now also be used for disaster recovery of multitenant container databases (CDBs). Other improvements include configuration health checks, resumable switchover operations, streamlined role transitions, support for the cascaded standby configurations, and user configurable thresholds regarding SLAs for recovery to the supported features.

Far Synch
: Enables zero-loss primary and standby Active Data Guard configurations that are separated by distances that can span continents.

Oracle Flex Clusters
: Large Oracle RAC Clusters deployed in a hub and spoke architecture while leveraging Flex ASM.

Point-in-time Recovery enhancements
: Recovery of tables and partitions to a specified point in time is possible using RMAN backups.

Chapter 12: Oracle and Hardware Architecture

This chapter was extensively rewritten in this edition to cover the emergence of growing memory footprints in platforms and the impact of engineered systems, especially the Oracle Exadata Database Machine, Database Appliance, SuperCluster, and Exalogic.

Chapter 13: Oracle Distributed Databases and Distributed Data

As Advanced Replication, AQ, and Streams are depreciated, much of the focus of this chapter is now on the role of two complementary middleware offerings for distributed database deployment: Tuxedo and GoldenGate.

Chapter 14: Oracle Extended Datatypes

This chapter describes capabilities beyond Oracle's standard set of datatypes. New features enhancing Oracle's support of extended datatypes include:

DICOM enhancements
: Protocol for exchange of DICOM images is supported.

XML DB integration
: XML DB installation is mandatory and the Database is XML aware.

XQuery enhancements
: XQuery updates, XQuery Full Text, and XQuery API for Java are supported.

Oracle Spatial and Graph enhancements
: Support for NURBS was added.

Chapter 15: Oracle and the Cloud

This chapter covers a variety of cloud deployment models where Oracle might be deployed leveraging capabilities described earlier in the book. This is the first edition of this book covering cloud computing.

APPENDIX B
Additional Resources

In this concise volume, we have attempted to give you a firm grounding in all the basic concepts you need to understand Oracle and use it effectively. We hope we have accomplished this goal. At the same time, we realize that there is more to using a complex product such as Oracle than simply understanding how and why it works the way it does. Although you can't use Oracle without a firm grasp of the foundations of the product, you will still need details if you're actually going to implement a successful system.

This appendix lists two types of additional sources of information for the topics covered in this book—relevant web sites, which act as a constantly changing resource for a variety of information, and a chapter-by-chapter list of relevant books, articles, and Oracle documentation.

For the chapter-by-chapter list, the sources fall into two basic categories: Oracle documentation/whitepapers and third-party sources. Typically, the Oracle documentation provides the type of hands-on information you will need regarding syntax and keywords, and while the whitepapers and third-party sources cover the topics in a more general and problem-solving way. We have listed the third-party sources first and ended each listing with the relevant Oracle documentation and whitepapers. Also note that some of the volumes listed here include previous Oracle release names in their titles. You can assume that by the time you are reading this, similar volumes exist (or will soon exist) for whatever version of Oracle you may be using (for example, Oracle Database 12*c*).

Web Sites

Oracle Corporation

Oracle's home page, featuring the latest company and marketing information, as well as some good technical and packaging information.

Oracle Technology Network [also accessible here]
: The focal point of Oracle Corporation's information intended for an audience of developers and administrators. You can find tons of stuff at the Oracle Technology Network (OTN), including low-cost developer versions or free downloads of most Oracle software and lots of information and discussion forums.

International Oracle Users Group (IOUG)
: The International Oracle Users Group web site includes information on meetings, links to Oracle resources, a technical repository, discussion forums, and special interest groups.

OraPub, Inc.
: Craig Shallahamer's site devoted to all things Oracle. Craig was a long-time Oracle employee in the performance analysis group and technical reviewer for various editions of this book.

O'Reilly Media, Inc.
: The O'Reilly web site that contains web pages for O'Reilly books, including links to Safari Books Online, and a variety of other helpful information.

Books and Oracle Documentation

The following books and Oracle documentation provide additional information for each chapter of this book.

Chapter 1: Introducing Oracle

Bryla, Bob, and Kevin Loney. *Oracle Database 11g DBA Handbook*. New York, NY: McGraw-Hill Oracle Press, 2007.

Ellison, Lawrence. *Oracle Overview and Introduction to SQL*. Belmont, CA: Oracle Corporation, 1985.

Greenwald, Rick et al. *Professional Oracle Programming*, Indianapolis, IN: Wrox/John Wiley & Sons, 2005.

Kelly, David. *Oracle Celebrates 30 Years of Innovation*, Oracle Magazine, July / August 2007.

Ralston, Anthony, ed. *Encyclopedia of Computer Science and Engineering*. New York, NY: Nostrand Reinhold Company, 1983.

Getting Started with Oracle NoSQL Database (available as online documentation). Redwood Shores, CA: Oracle Corporation, 2013.

Oracle Database 12c Product Family (An Oracle White Paper). Redwood Shores, CA: Oracle Corporation, 2013.

Oracle Database Concepts 12c Release 1 (12.1). Redwood Shores, CA: Oracle Corporation, 2013.

Oracle Database New Features Guide 12c Release 1 (12.1). Redwood Shores, CA: Oracle Corporation, 2013.

Oracle Database SQL Language Reference 12c Release 1 (12.1). Redwood Shores, CA: Oracle Corporation, 2013.

Oracle In-Memory Database Cache Users Guide 11g Release 2 (11.2.2). Redwood Shores, CA: Oracle Corporation, 2013.

Oracle MySQL Reference Manual 5.6 (available as online documentation). Redwood Shores, CA: Oracle Corporation, 2013.

Plug into the Cloud with Oracle Database 12c (An Oracle White Paper). Redwood Shores, CA: Oracle Corporation, 2013.

Chapter 2: Oracle Architecture

Loney, Kevin. *Oracle Database 11g The Complete Reference*. New York, NY: McGraw-Hill, 2008.

Oracle Database Concepts 12c Release 1 (12.1). Redwood Shores, CA: Oracle Corporation, 2013.

Oracle Database Reference 12c Release 1 (12.1). Redwood Shores, CA: Oracle Corporation, 2013.

Chapter 3: Installing and Running Oracle

Oracle Database Concepts 12c Release 1 (12.1). Redwood Shores, CA: Oracle Corporation, 2013.

Oracle Database Installation Guide 12c Release 1 (12.1) for Microsoft Windows. Redwood Shores, CA: Oracle Corporation, 2013.

Oracle Database Installation Guide 12c Release 1 (12.1) for Linux. Redwood Shores, CA: Oracle Corporation, 2013.

Oracle Database Installation Guide 12c Release 1 (12.1) for Oracle Solaris. Redwood Shores, CA: Oracle Corporation, 2013.

Oracle Database Net Services Administrators Guide 12c Release 1 (12.1). Redwood Shores, CA: Oracle Corporation, 2013.

Oracle Enterprise Manager Cloud Control Basic Installation Guide 12c Release 2 (12.1.0.2). Redwood Shores, CA: Oracle Corporation, 2013.

Chapter 4: Data Structures

Date, C.J., *Relational Theory for Computer Professionals*. Sebastopol, CA: O'Reilly Media, Inc., 2013.

Date, C.J., *SQL and Relational Theory*. Sebastopol, CA: O'Reilly Media, Inc., 2011.

Feuerstein, Steven, with Bill Pribyl. *Oracle PL/SQL Programming*, Fifth Edition. Sebastopol, CA: O'Reilly Media, Inc., 2009.

Oracle Database Concepts 12c Release 1 (12.1). Redwood Shores, CA: Oracle Corporation, 2013.

Oracle Database SQL Tuning Guide 12c Release 1 (12.1). Redwood Shores, CA: Oracle Corporation, 2013.

Chapter 5: Managing Oracle

Himatsingka, Bhaskar, and Juan Loaiza. "How to Stop Defragmenting and Start Living: The Definitive Word on Fragmentation." Paper no. 711. Belmont, CA: Oracle Corporation, 1998.

Oracle Automatic Storage Management Administrator's Guide. Redwood Shores, CA: Oracle Corporation, 2013.

Oracle Database Administrator's Guide 12c Release 1 (12.1). Redwood Shores, CA: Oracle Corporation, 2013.

Oracle Database Backup and Recovery User's Guide 12c Release 1 (12.1). Redwood Shores, CA: Oracle Corporation, 2013.

Oracle Database Concepts 12c Release 1 (12.1). Redwood Shores, CA: Oracle Corporation, 2013.

Oracle Enterprise Manager Cloud Control Administrator's Guide 12c Release 3 (12.1.0.3). Redwood Shores, CA: Oracle Corporation, 2013.

Oracle Enterprise Manager Cloud Control Basic Installation Guide 12c Release 2 (12.1.0.2). Redwood Shores, CA: Oracle Corporation, 2013.

Oracle Enterprise Manager Lifecycle Management Administrator's Guide 12c Release 3 (12.1.0.3). Redwood Shores, CA: Oracle Corporation, 2013.

Chapter 6: Oracle Security, Auditing, and Compliance

Knox, David et al. *Applied Oracle Security*. New York, NY: McGraw-Hill Oracle Press, 2009.

Feurstein Steven, and Bill Pribyl. *Oracle PL/SQL Programming*. Sebastopol, CA: O'Reilly Media, Inc., 2005.

Nanda, Arup, and Steven Feuersten. *Oracle PL/SQL for DBAs*. Sebastopol, CA: O'Reilly Media, Inc., 2005.

Oracle Database Security Guide 12c Release 1 (12.1). Redwood Shores, CA: Oracle Corporation, 2013.

Oracle Database Advanced Security Guide 12c Release 1 (12.1). Redwood Shores, CA: Oracle Corporation, 2013.

Oracle Database Enterprise User Security Administrator's Guide 12c Release 1 (12.1). Redwood Shores, CA: Oracle Corporation, 2013.

Oracle Database Label Security Administrator's Guide 12c Release 1 (12.1). Redwood Shores, CA: Oracle Corporation, 2013.

Oracle Database Vault Administrator's Guide 12c Release 1 (12.1). Redwood Shores, CA: Oracle Corporation, 2013.

Oracle Database 2 Day + Security Guide 12c Release 1 (12.1). Redwood Shores, CA: Oracle Corporation, 2013.

Chapter 7: Oracle Performance

Niemiec, Rich. *Oracle Database 11g Release 2 Performance Tuning Tips & Techniques*. New York, NY: McGraw-Hill, 2012.

Oracle Database Concepts 12c Release 1 (12.1). Redwood Shores, CA: Oracle Corporation, 2013.

Oracle Database Performance Tuning Guide 12c Release 1 (12.1). Redwood Shores, CA: Oracle Corporation, 2013.

Oracle Database SQL Tuning Guide 12c Release 1 (12.1). Redwood Shores, CA: Oracle Corporation, 2013.

Oracle Database 2 Day + Performance Tuning Guide 12c Release 1 (12.1). Redwood Shores, CA: Oracle Corporation, 2013.

Chapter 8: Oracle Multiuser Concurrency

Oracle Database Concepts 12c Release 1 (12.1). Redwood Shores, CA: Oracle Corporation, 2013.

Chapter 9: Oracle and Transaction Processing

Gray, Jim, and Andreas Reuter. *Transaction Processing: Concepts and Techniques*. San Francisco, CA: Morgan Kaufmann Publishers, 1992.

Edwards, Jeri, with Deborah DeVoe. *3-Tier Client/Server at Work*. New York, NY: John Wiley & Sons, 1997.

Oracle Database Development Guide 12c Release 1 (12.1). Redwood Shores, CA: Oracle Corporation, 2013.

Oracle Database Concepts 12c Release 1 (12.1). Redwood Shores, CA: Oracle Corporation, 2013.

Oracle Database Java Developer's Guide 12c Release 1 (12.1). Redwood Shores, CA: Oracle Corporation, 2013.

Oracle Database Net Services Administrators Guide 12c Release 1 (12.1). Redwood Shores, CA: Oracle Corporation, 2013.

Oracle Database 2 Day + Real Application Clusters Guide 12c Release 1 (12.1). Redwood Shores, CA: Oracle Corporation, 2013.

Chapter 10: Oracle Data Warehousing and Business Intelligence

Inmon, W.H. *Building the Data Warehouse*. New York, NY: John Wiley & Sons, 2005.

Kimball, Ralph, and Margy Ross. *The Data Warehouse Lifecycle Toolkit*. New York, NY: John Wiley & Sons, 2013.

Linoff, Gordon, and Michael Berr. *Data Mining Techniques*. New York, NY: John Wiley & Sons, 2011.

Miller, Simon, and William Hutchinson. *Oracle Business Intelligence Applicaitons*. New York, NY: McGraw-Hill Oracle Press, 2013.

Peppers, Don, and Martha Rogers. *Enterprise One to One*. New York, NY: Currency Doubleday, 1997.

Peppers, Don, Martha Rogers, and Bob Dorf. *One to One Fieldbook*. New York, NY: Currency Doubleday, 1999.

Plunkett, Tom et al. *Oracle Big Data Handbook*. New York, NY: McGraw-Hill Oracle Press, 2013.

Schrader, Michael et al. *Oracle Essbase & Oracle OLAP*. New York, NY: McGraw-Hill Oracle Press, 2010.

Stackowiak, Robert et al. *Oracle Data Warehousing and Business Intelligence Solutions*. Indianapolis, IN: John Wiley & Sons, 2007.

Stackowiak, Robert. "Why Bad Data Warehouses Happen to Good People." *The Journal of Data Warehousing*, April 1997.

White, Tom. *Hadoop: The Definitive Guide*. Sebastopol, CA: O'Reilly Media, 2011.

Oracle Big Data Appliance Software User's Guide Release 2 (2.2). Redwood Shores, CA: Oracle Corporation, 2013.

Oracle Data Mining Concepts 12c Release 1 (12.1). Redwood Shores, CA: Oracle Corporation, 2013.

Oracle Database Concepts 12c Release 1 (12.1). Redwood Shores, CA: Oracle Corporation, 2013.

Oracle Database Data Warehousing Guide 12c Release 1 (12.1). Redwood Shores, CA: Oracle Corporation, 2013.

Oracle Exalytics In-Memory Machine: A Brief Introduction (An Oracle White Paper). Redwood Shores, CA: Oracle Corporation, 2013.

Oracle OLAP User's Guide 12c Release 1 (12.1). Redwood Shores, CA: Oracle Corporation, 2013.

Oracle R Enterprise User's Guide 12c Release 1.2 for Linux, Solaris, AIX, and Windows. Redwood Shores, CA: Oracle Corporation, 2013.

Chapter 11: Oracle and High Availability

Chen, Lee et al. "RAID: High Performance, Reliable Secondary Storage." *ACM Computing Surveys*, June 1994.

Oracle Database Backup and Recovery User's Guide 12c Release 1 (12.1). Redwood Shores, CA: Oracle Corporation, 2013.

Oracle Database Concepts 12c Release 1 (12.1). Redwood Shores, CA: Oracle Corporation, 2013.

Oracle Database Global Data Services Concepts and Administration 12c Release 1 (12.1). Redwood Shores, CA: Oracle Corporation, 2013.

Oracle Database 2 Day + Real Application Clusters Guide 12c Release 1 (12.1). Redwood Shores, CA: Oracle Corporation, 2013.

Oracle Data Guard Concepts and Administration 12c Release 1 (12.1). Redwood Shores, CA: Oracle Corporation, 2013.

Oracle High Availability Overview 12c Release 1 (12.1). Redwood Shores, CA: Oracle Corporation, 2013.

Chapter 12: Oracle and Hardware Architecture

Greenwald, Rick et al. *Achieving Extreme Performance with Oracle Exadata*. New York, NY: McGraw-Hill Oracle Press, 2011.

Morse, H. Stephen. *Practical Parallel Computing*. Cambridge, MA: AP Professional, 1994.

Pfister, Gregory. *In Search of Clusters*. Upper Saddle River, NJ: Prentice Hall PTR, 1995.

A Technical Overview of the Oracle Exadata Database Machine and Exadata Storage Server (An Oracle White Paper). Redwood Shores, CA: Oracle Corporation, 2012.

Oracle Database Appliance Owner's Guide Release 2.6 for Linux x86-64. Redwood Shores, CA: Oracle Corporation, 2013.

Oracle Database Appliance X3-2 (An Oracle White Paper). Redwood Shores, CA: Oracle Corporation, 2013.

Oracle Exalogic Elastic Cloud Machine Owner's Guide Release EL X2-2 and EL X3-2. Redwood Shores, CA: Oracle Corporation, 2013.

Oracle SuperCluster T5-8: Servers, Storage, Networking, and Software – Optimized and Ready to Run (An Oracle White Paper). Redwood Shores, CA: Oracle Corporation, 2013.

Chapter 13: Oracle Distributed Databases and Distributed Data

Oracle Database Concepts 12c Release 1 (12.1). Redwood Shores, CA: Oracle Corporation, 2013.

Oracle Database Global Data Services Concepts and Administration Guide 12c Release 1 (12.1). Redwood Shores, CA: Oracle Corporation, 2013.

Oracle GoldenGate Windows and Unix Administrator's Guide 11g Release 2 Patch Set 1 (11.2.0.1). Redwood Shores, CA: Oracle Corporation, 2012.

Oracle Tuxedo Product Overview 12c Release 1 (available as online documentation). Redwood Shores, CA: Oracle Corporation, 2012.

Chapter 14: Oracle Extended Datatypes

Oracle Database Concepts 12c Release 1 (12.1). Redwood Shores, CA: Oracle Corporation, 2013.

Oracle Database Java Developer's Guide 12c Release 1 (12.1). Redwood Shores, CA: Oracle Corporation, 2013.

Oracle Database Object Relational Developer's Guide 12c Release 1 (12.1). Redwood Shores, CA: Oracle Corporation, 2013.

Oracle Database SecureFiles and Large Objects Developer's Guide 12c Release 1 (12.1). Redwood Shores, CA: Oracle Corporation, 2013.

Oracle Multimedia User's Guide 12c Release 1 (12.1). Redwood Shores, CA: Oracle Corporation, 2013.

Oracle Multimedia DICOM Developer's Guide 12c Release 1 (12.1). Redwood Shores, CA: Oracle Corporation, 2013.

Oracle Multimedia Reference 12c Release 1 (12.1). Redwood Shores, CA: Oracle Corporation, 2013.

Oracle Spatial and Graph Developer's Guide 12c Release 1 (12.1). Redwood Shores, CA: Oracle Corporation, 2013.

Oracle Spatial and Graph GeoRaster Developer's Guide 12c Release 1 (12.1). Redwood Shores, CA: Oracle Corporation, 2013.

Oracle Text Reference 12c Release 1 (12.1). Redwood Shores, CA: Oracle Corporation, 2013.

Oracle XML DB Developer's Guide 12c Release 1 (12.1). Redwood Shores, CA: Oracle Corporation, 2013.

Chapter 15: Oracle and the Cloud

Oracle Application Express Administrator's Guide Release 4.2 for Oracle Database 12c. Redwood Shores, CA: Oracle Corporation, 2013.

Oracle Application Express Application Builder User's Guide Release 4.2 for Oracle Database 12c. Redwood Shores, CA: Oracle Corporation, 2013.

Oracle Database Concepts 12c Release 1 (12.1). Redwood Shores, CA: Oracle Corporation, 2013.

Oracle Enterprise Manager Cloud Administration Guide 12c Release 3 (12.1.0.3). Redwood Shores, CA: Oracle Corporation, 2013.

Oracle Enterprise Manager Lifecycle Management Administrator's Guide 12c Release 3 (12.1.0.3). Redwood Shores, CA: Oracle Corporation, 2013.

Plug into the Cloud with Oracle Database 12c (An Oracle White Paper). Redwood Shores, CA: Oracle Corporation, 2013.

Index

Symbols

A

We'd like to hear your suggestions for improving our indexes. Send email to index@oreilly.com.

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z

About the Authors

Rick Greenwald has been active in the world of computer software for over two decades, including stints with Data General, Cognos, and Gupta. He is currently Director of Product Management with Oracle Corporation, responsible for the Oracle Database Cloud. He has been a principal author of 20 books and countless articles on a variety of technical topics, and has spoken at conferences and training sessions across six continents. Rick's titles include five editions of *Oracle Essentials* (principal author with Robert Stackowiak and Jonathan Stern, O'Reilly Media, Inc), a number of books on Application Express and its predecessors, and books on data warehousing, Oracle programming, and Exadata.

Robert Stackowiak has more than 25 years of experience in data warehousing and business intelligence architecture, software development, new database and systems product introduction, and technical sales and sales consulting. During the writing of this edition of *Oracle Essentials*, he is Vice President of Information Architecture and Big Data in Oracle's Enterprise Solutions Group. He has spoken on related Oracle topics at a wide variety of conferences, including Gartner Group's BI Summit, TDWI, ComputerWorld's BI & Analytics Perspectives, Oracle OpenWorld, and numerous IOUG events. Among the other books he co-authored are the following: *Achieving Extreme Performance with Oracle Exadata* (McGraw-Hill Oracle Press), *Professional Oracle Programming* (WROX), and *Oracle Data Warehousing and Business Intelligence Solutions* (Wiley). He can be followed on Twitter @rstackow.

Jonathan Stern used more than 13 years of experience in contributing to the original edition of this book. His background included senior positions in consulting, systems architecture, and technical sales at Oracle and other companies. He demonstrated in-depth knowledge of the Oracle Database across major open systems hardware and operating systems. His expertise included database tuning, scaling, parallelism, clustering, and high availability for transaction processing systems and data warehousing implementations. Some of the fundamentals of Oracle that he originally contributed to the first edition live on in this edition of *Oracle Essentials*, though much has changed in the Oracle Database since Jonathan passed on. Each edition of this book is a reminder of the talent and attention to detail that he possessed.

Colophon

The animals on the cover of *Oracle Essentials*, Fifth Edition are cicadas. There are about 1,500 species of cicada. In general, cicadas are large insects with long thin wings that are perched above an inch-long abdomen. Their heads are also large and contain three eyes and a piercing and sucking mechanism with which to extrude sap from trees. Cicadas are known for their characteristic shrill buzz, which is actually the male's mating song, one of the loudest known insect noises.

Cicadas emerge from the ground in the spring or summer, molt, then shed their skin in the form of a shell. They stay near trees and plants, where they live for four to six weeks with the sole purpose of mating. The adult insects then die, and their young hatch and burrow into the ground. They attach to tree roots and feed off the sap for 4 to 17 years, after which time they emerge and continue the mating cycle. Cicadas have one of the longest life spans of any insect; the most common species is the periodical cicada, which lives underground for 13 to 17 years.

The cover image is an original 19th-century engraving from *Cuvier's Animals*. The cover font is Adobe ITC Garamond. The text font is Adobe Minion Pro; the heading font is Adobe Myriad Condensed; and the code font is Dalton Maag's Ubuntu Mono.