To Our Valued Readers:

In a CertCities.com article dated December 15, 2001, Oracle certification was ranked #2 in a list of the “10 Hottest Certifications for 2002.” This shouldn’t come as a surprise, especially when you consider the fact that the OCP program nearly tripled in size (from 30,000 to 80,000) in the last year. Oracle continues to expand its dominance in the database market, and as companies begin integrating Oracle9i systems into their IT infrastructure, you can be assured of high demand for professionals with the Oracle Certified Associate and Oracle Certified Professional certifications.

Sybex is proud to have helped thousands of Oracle certification candidates prepare for the exams over the years, and we are excited about the opportunity to continue to provide professionals like you with the skills needed to succeed in the highly competitive IT industry.

Our authors and editors have worked hard to ensure that the Oracle9i Study Guide you hold in your hands is comprehensive, in-depth, and pedagogically sound. We’re confident that this book will meet and exceed the demanding standards of the certification marketplace and help you, the Oracle9i certification candidate, succeed in your endeavors.

Good luck in pursuit of your Oracle9i certification!

Neil Edde
Associate Publisher—Certification
Sybex, Inc.
To Brenda and Emily, the two most important people in my life.

—Joseph C. Johnson
Acknowledgments

I would like to acknowledge the many people who helped shape the professional experiences that led to my writing of this book. This includes my Oracle colleagues at Greenbrier & Russel, Inc., whose Oracle challenges have also helped shape my own: Christopher Guillaume, John Golden, Jesus Quinones, Jim Whittle, and Jane Yang.

Included too are the clients and fellow DBAs who have offered me many interesting Oracle experiences in recent years, including: Jonathan Jackson, Tim Eleyet, Linda Ramme, Bob Budden, Greg Van Haren and John Aaholm at American Transmission Company; Tony Gastel and Audrey Smith at Highland Community College; George Carpenter and Lorin Weiler at Dean Health Plan; Mike Gavin, Jerri Wang, Dan Hartsman, Thomas Baas, and Ted Drankus at Milwaukee Area Technical College; and Jim Schwarz and Reginald Bron at Snap-On Tools.

I’d like to thank Matt Weishan for writing or revising all of the questions in the 9i book and on the CD. I’m glad we were able to work together again on this project.

Thanks also to Bob Bryla and Ashok Hanumanth for their efforts during the technical reviews, and all the people at Sybex who made this project a smooth one: Christine McGeever, Jeff Kellum, Dan Schiff, Linda Stephenson, and Leslie Light.

Finally, I say thank you to my wife, Brenda, who somehow always found the time to help me by proofreading chapters or fixing up screen shots whenever deadlines were looming.
Introduction

There is high demand for professionals in the information technology (IT) industry, and Oracle certifications are the hottest credential in the database world. You have made the right decision to pursue certification, because being Oracle certified will give you a distinct advantage in this highly competitive market.

Many readers may already be familiar with Oracle and do not need an introduction to the Oracle database world. For those who aren’t familiar with the company, Oracle, founded in 1977, sold the first commercial relational database and is now the world’s leading database company and second-largest independent software company, with revenues of more than $10 billion, serving more than 145 countries.

Oracle databases are the de facto standard for large Internet sites, and Oracle advertisers are boastful but honest when they proclaim, “The Internet Runs on Oracle.” Almost all big Internet sites run Oracle databases. Oracle’s penetration of the database market runs deep and is not limited to dot-com implementations. Enterprise resource planning (ERP) application suites, data warehouses, and custom applications at many companies rely on Oracle. The demand for DBA resources remains higher than others during weak economic times.

This book is intended to help you on your exciting path toward becoming an Oracle Certified Professional (OCP) and Oracle Certified Master (OCM). Basic knowledge of Oracle SQL is an advantage when reading this book but is not mandatory. Using this book and a practice database, you can start learning Oracle and pass the 1Z0-033 test: Oracle9i Database: Performance Tuning.

Why Become an Oracle Certified Professional?

The number one reason to become an OCP is to gain more visibility and greater access to the industry’s most challenging opportunities. Oracle certification is the best way to demonstrate your knowledge and skills in Oracle database systems. The certification tests are scenario-based, which is the most effective way to assess your hands-on expertise and critical problem-solving skills.

Certification is proof of your knowledge and shows that you have the skills required to support Oracle core products. The Oracle certification
program can help a company identify proven performers who have demonstrated their skills and who can support the company’s investment in Oracle technology. It demonstrates that you have a solid understanding of your job role and the Oracle products used in that role.

OCPs are among the best paid in the IT industry. Salary surveys consistently show the OCP certification to yield higher salaries than other certifications, including Microsoft, Novell, and Cisco.

So, whether you are beginning a career, changing careers, securing your present position, or seeking to refine and promote your position, this book is for you!

**Oracle Certifications**

Oracle certifications follow a track that is oriented toward a job role. There are database administration, database operator, and developer tracks. Within each track, Oracle has a three-tiered certification program:

- The first tier is the Oracle Certified Associate (OCA). OCA certification typically requires you to complete two exams, the first via the Internet and the second in a proctored environment.

- The next tier is the Oracle Certified Professional (OCP), which builds upon and requires an OCA certification. The additional requirements for OCP certification are additional proctored exams.

- The third and highest tier is the Oracle Certified Master (OCM). OCM certification builds upon and requires OCP certification. To achieve OCM certification, you must attend two advanced Oracle Education classroom courses (from a specific list of qualifying courses) and complete a practicum exam.

The following material will address only the database administration track, because at the time of this writing, it was the only 9i track offered by Oracle. The other tracks have 8 and 8i certifications and will undoubtedly have 9i certifications. See the Oracle website at [http://www.oracle.com/education/certification/](http://www.oracle.com/education/certification/) for the latest information.

**Oracle9i Certified Database Associate**

The role of the database administrator (DBA) has become a key to success in today’s highly complex database systems. The best DBAs work behind the scenes, but are in the spotlight when critical issues arise. They plan, create, maintain, and ensure that the database is available for the business. They
are always watching the database for performance issues and to prevent unscheduled downtime. The DBA’s job requires broad understanding of the architecture of an Oracle database and expertise in solving problems.

The Oracle9i Certified Database Associate is the entry-level certification for the database administration track and is required to advance toward the more senior certification tiers. This certification requires you to pass two exams that demonstrate your knowledge of Oracle basics:

- 1Z0-007: Introduction to Oracle9i: SQL
- 1Z0-031: Oracle9i Database: Fundamentals I

The 1Z0-007 exam, Introduction to Oracle9i: SQL, is offered on the Internet. The 1Z0-031 exam, Oracle9i Database: Fundamentals I, is offered at a Sylvan Prometric facility.

**Oracle9i Certified Database Administrator (OCP)**

The OCP tier of the database administration track challenges you to demonstrate your continuing experience and knowledge of Oracle technologies. The Oracle9i Certified Database Administrator certification requires achievement of the Certified Database Associate tier, as well as passing the following two exams at a Sylvan Prometric facility:

- 1Z0-032: Oracle9i Database: Fundamentals II
- 1Z0-033: Oracle9i Database: Performance Tuning

**Oracle9i Certified Master**

The Oracle9i Certified Master is the highest level of certification that Oracle offers. To become a certified master, you must first achieve OCP status, then complete two advanced instructor-led classes at an Oracle education facility, and finally pass a hands-on exam at Oracle Education. The classes and practicum exam are offered only at an Oracle education facility and may require travel. The advanced classes that will count toward your OCM requirement include the following:

- Oracle9i: Program with PL/SQL
- Oracle9i: Advanced PL/SQL
- Oracle9i: SQL Tuning Workshop
- Oracle9i: High Availability in an Internet Environment
Introduction

- Oracle9i: Database: Implement Partitioning
- Oracle9i: Real Application Clusters Implementation
- Oracle9i: Data Warehouse Administration
- Oracle9i: Advanced Replication
- Oracle9i: Enterprise Manager

More Information

Current information about Oracle certification can be found at http://www.oracle.com/education/certification. Follow the Certification link and choose the track that you are interested in. Read the Candidate Guide for the test objectives and test contents, and keep in mind that they can change at any time without notice.

OCA/OCP Study Guides

The Oracle9i database administration track certification consists of four tests: two for OCA level and two more for OCP level. Sybex offers several study guides to help you achieve this certification:

- OCA/OCP: Introduction to Oracle9i™ SQL Study Guide (exam 1Z0-007: Introduction to Oracle9i: SQL)
- OCA/OCP: Oracle9i™ DBA Fundamentals I Study Guide (exam 1Z0-031: Oracle9i Database: Fundamentals I)
- OCP: Oracle9i™ DBA Fundamentals II Study Guide (exam 1Z0-032: Oracle9i Database: Fundamentals II)
- OCP: Oracle9i™ DBA Performance Tuning (exam 1Z0-033: Oracle9i Database: Performance Tuning)

Additionally, these four books are offered in a boxed set: OCP: Oracle9i™ DBA Certification Kit.

Skills Required for DBA Certification

To pass the certification exams, you need to master the following skills:

- Write SQL SELECT statements that display data from either single or multiple tables.
- Restrict, sort, aggregate, and manipulate data using both single and group functions.
• Create and manage tables, views, constraints, synonyms, sequences, and indexes.
• Create users and roles to control user access and maintain security.
• Understand Oracle Server architecture (database and instance).
• Understand the physical and logical storage of the database, and be able to manage space allocation and growth.
• Manage data, including its storage, loading, and reorganization.
• Manage redo logs, automatic undo, and rollback segments.
• Use globalization features to choose a database character set and National Language Support (NLS) parameters.
• Configure Oracle Net on the server side and the client side.
• Use backup and recovery options.
• Archive redo log files and hot backups.
• Perform backup and recovery operations using Recovery Manager (RMAN).
• Use data dictionary views and set database parameters.
• Configure and use Shared Server and Connection Manager.
• Identify and tune database and SQL performance.
• Use the tuning/diagnostics tools STATSPACK, TKPROF, and EXPLAIN PLAN.
• Tune the size of data blocks, the shared pool, the buffer caches, and rollback segments.
• Diagnose contention for latches, locks, and rollback segments.

Tips for Taking the OCP Exam
Use the following tips to help you prepare for and pass each exam:

• Each OCP test contains about 60–80 questions to be completed in 90 minutes. Answer the questions you know first, so that you do not run out of time.
• Many questions on the exam have answer choices that at first glance look identical. Read the questions carefully. Do not just jump to conclusions. Make sure that you clearly understand exactly what each question asks.

• Most of the test questions are scenario-based. Some of the scenarios contain nonessential information and exhibits. You need to be able to identify what’s important and what’s not important.

• Do not leave any questions unanswered. There is no negative scoring. After selecting an answer, you can mark a difficult question or one that you’re unsure of and come back to it later.

• When answering questions that you are not sure about, use a process of elimination to get rid of the obviously incorrect answers first. Doing this greatly improves your odds if you need to make an educated guess.

• If you’re not sure of your answer, mark it for review and then look for other questions that may help you eliminate any incorrect answers. At the end of the test, you can go back and review the questions that you marked for review.

Where Do You Take the Exam?

You take the Introduction to Oracle9i: SQL exam (1Z0-007) via the Internet. To register for an online Oracle certification exam, you will need an Internet connection of at least 33Kbps, but a 56Kbps, LAN, or broadband connection is recommended. You will also need either Internet Explorer 5.0 (or above) or Netscape 4.x (Oracle does not recommend Netscape 5.x or 6.x). At the time of this writing, the online 1Z0-007 exam is $90. If you do not have a credit card to use for payment, you will need to contact Oracle to purchase a voucher. You can pay with a certification voucher, promo codes, or credit card.

You may take the other exams at any of the more than 800 Sylvan Prometric Authorized Testing Centers around the world. For the location of a testing center near you, call 1-800-891-3926. Outside the United States and Canada, contact your local Sylvan Prometric Registration Center. Usually, the tests can be taken in any order.

To register for a proctored Oracle Certified Professional exam at a Sylvan Prometric test center:

• Determine the number of the exam you want to take.
- Register with Sylvan Prometric online at http://www.2test.com or in North America, by calling 1-800-891-EXAM (800-891-3926). At this point, you will be asked to pay in advance for the exam. At the time of this writing, the exams are $125 each and must be taken within one year of payment.

- When you schedule the exam, you'll get instructions regarding all appointment and cancellation procedures, the ID requirements, and information about the testing-center location.

You can schedule exams up to six weeks in advance or as soon as one working day before the day you wish to take it. If something comes up and you need to cancel or reschedule your exam appointment, contact Sylvan Prometric at least 24 hours in advance.

**What Does This Book Cover?**

This book covers everything you need to pass the Oracle9i: Performance Tuning exam. This exam is part of the Oracle9i Certified Professional certification tier in the database administration track. It teaches you how to use Oracle’s tuning methodologies and techniques to improve database performance. Each chapter begins with a list of exam objectives.

**Chapter 1** Begins with an overview of Oracle’s performance tuning methodologies and review of the Oracle architecture.

**Chapter 2** Discusses sources of performance tuning information, from data dictionary and dynamic performance views to Oracle-supplied tuning utilities.

**Chapter 3** Discusses how to measure and tune application design in terms of optimizer behavior, execution plans, and indexing strategies.

**Chapter 4** Explains how to measure and improve the performance of the Shared Pool.

**Chapter 5** Explores how to measure and improve the performance of the Database Buffer Cache.

**Chapter 6** Discusses how to measure and tune the performance of other SGA areas like the Large Pool, Java Pool, and Shared Server processes.

**Chapter 7** Explains how to monitor and manage the performance of Oracle redo mechanisms.
Chapter 8  Describes how to measure and manage the performance of the portions of the Oracle architecture that perform physical disk I/O.

Chapter 9  Explains how to measure and tune performance issues related to contention for latches, Free Lists, and locks.

Chapter 10  Discusses how Oracle’s Resource Manager features can be used to improve the utilization of Oracle Server resources such as CPU and memory.

Each chapter ends with Review Questions that are specifically designed to help you retain the knowledge presented. To really nail down your skills, read and answer each question carefully.

How to Use This Book

This book can provide a solid foundation for the serious effort of preparing for the OCP database administration exam track. To best benefit from this book, use the following study method:

1. Take the Assessment Test immediately following this introduction. (The answers are at the end of the test.) Carefully read over the explanations for any questions you get wrong, and note which chapters the material comes from. This information should help you plan your study strategy.

2. Study each chapter carefully, making sure that you fully understand the information and the test objectives listed at the beginning of each chapter. Pay extra close attention to any chapter related to questions you missed in the Assessment Test.

3. Complete all hands-on exercises in the chapter, referring to the chapter so that you understand the reason for each step you take. If you do not have an Oracle Database available, be sure to study the examples carefully. Answer the Review Questions related to that chapter. (The answers appear at the end of each chapter, after the “Review Questions” section.)

4. Note the questions that confuse or trick you, and study those sections of the book again.

5. Before taking the exam, try your hand at the two bonus Practice Exams that are included on the CD that comes with this book. The
questions on these exams appear only on the CD. This will give you a complete overview of what you can expect to see on the real test.

6. Remember to use the products on the CD included with this book. The electronic flashcards and the EdgeTest exam preparation software have been specifically designed to help you study for and pass your exam. The electronic flashcards can be used on your Windows computer or on your Palm device.

To learn all the material covered in this book, you'll need to apply yourself regularly and with discipline. Try to set aside the same time period every day to study, and select a comfortable and quiet place to do so. If you work hard, you will be surprised at how quickly you learn this material. All the best!

What’s on the CD?

We have worked hard to provide some really great tools to help you with your certification process. All of the following tools should be loaded on your workstation when you’re studying for the test.

**The EdgeTest for Oracle Certified DBA Preparation Software**

Provided by EdgeTek Learning Systems, this test-preparation software prepares you to pass the Oracle9i Database: Performance Tuning exam. In this test, you will find all of the questions from the book, plus two bonus Practice Exams that appear exclusively on the CD. You can take the Assessment Test, test yourself by chapter, take one or both of the Practice Exams, or take an exam randomly generated from all of the questions.

**Electronic Flashcards for PC and Palm Devices**

After you read the *OCP: Oracle9i Performance Tuning Study Guide*, read the Review Questions at the end of each chapter, and study the Practice Exams included on the CD. But wait, there’s more! Test yourself with the flashcards included on the CD. If you can get through these difficult questions and understand the answers, you’ll know that you’re ready for the exam.

The flashcards include over 150 questions specifically written to hit you hard and make sure you are ready for the exam. Between the Review Questions, Practice Exams, and flashcards, you should be more than prepared for the exam.
OCP: Oracle9i Performance Tuning Study Guide in PDF
Sybex is now offering this Oracle certification book on the CD so you can read the book on your PC or laptop. It is in Adobe Acrobat format. Acrobat Reader 5 is also included on the CD. This will be extremely helpful to readers who fly or commute on a bus or train and don’t want to carry a book, as well as to readers who find it more comfortable reading from their computer.

How to Contact the Author
You can reach Joe Johnson via e-mail at jjohnson@gr.com.
Assessment Test

1. Certain types of operating system parameters are tunable by the DBA. What are these types of parameters called?
   A. Kernel parameters
   B. Base parameters
   C. Initialization parameters
   D. Elementary parameters

2. Latches protect access to what Oracle structures?
   A. Datafiles
   B. Control files
   C. Tables
   D. Memory structures

3. Which of the following is TRUE when you use the EXPLAIN PLAN FOR... syntax to generate an Explain Plan for an SQL statement using SQL*Plus?
   A. The query itself is not executed. Only the Explain Plan is created.
   B. The information does not get stored in the PLAN_TABLE unless AUTOTRACE is turned on.
   C. The TKPROF utility automatically runs after the EXPLAIN PLAN FOR statement to generate formatted output.
   D. The query is executed and the Explain Plan is created.

4. The DBA wishes to make sure that certain users have access to all server resources without restriction. Which default resource group should the BDA put these users in?
   A. OTHER_GROUPS
   B. SYS_GROUP
   C. LOW_GROUP
   D. DEFAULT_CONSUMER_GROUP
5. The EMPTY_BLOCKS column in the DBA_TABLES view is a measure of what?
   A. The total number of empty blocks in the table
   B. The total number of empty blocks above the High Water Mark in the table
   C. The total number of empty blocks below the High Water Mark in the table
   D. None of the above

6. In the default Oracle9i configuration, what is the default method of client connections?
   A. Dedicated Server
   B. Shared Server
   C. Client/Server
   D. ODBC

7. The DBA and developer are building a student records system for a university. They decide to create a STUDENT_HISTORY table using the following syntax. How will this table be partitioned?

   SQL> CREATE TABLE student_history
             (student_id NUMBER(10),
              degree VARCHAR2(3),
              graduation_date DATE,
              final_gpa NUMBER)
             PARTITION BY RANGE (graduation_date)
             SUBPARTITION BY HASH(student_id) SUBPARTITIONS 4
             STORE IN (hist_tab01, hist_tab02, hist_tab03, hist_tab04)
             (PARTITION p_1997 VALUES LESS THAN (TO_DATE('01-JUN-1997','DD-MON-YYYY')),
              (TO_DATE('01-JUN-1998','DD-MON-YYYY')),
              (TO_DATE('01-JUN-1999','DD-MON-YYYY')),
              (TO_DATE('01-JUN-2000','DD-MON-YYYY')),
16 (TO_DATE('01-JUN-2000','DD-MON-YYYY')),
17 PARTITION p_2001 VALUES LESS THAN
18 (TO_DATE('01-JUN-2001','DD-MON-YYYY')),
19 PARTITION p_ERROR VALUES LESS THAN (MAXVALUE));

A. Hash
B. List
C. Range
D. Composite

8. How many of a database’s resource plans can be active at the instance-level at the same time?
   A. All of them
   B. None of them
   C. One of them
   D. Up to four

9. Your database is in ARCHIVELOG mode and a log switch occurs. The Online Redo Log being switched to has not yet been archived. What will happen?
   A. LGWR will overwrite the log.
   B. The instance will crash.
   C. Additional online logs will automatically be created to accommodate the log switch request.
   D. The LGWR process will wait for ARC0 to complete archiving of the Online Redo Log file.

10. Which of the following commands would not cause dirty blocks to be written by DBW0?
    A. ALTER TABLESPACE DATA BEGIN BACKUP
    B. SHUTDOWN IMMEDIATE
    C. SHUTDOWN ABORT
    D. ALTER SYSTEM SWITCH LOGFILE
11. Which of the following is *not* one of the two basic tenets of performance tuning?
   A. Adding memory to the Server
   B. Increasing the number of disks available to the application
   C. Decreasing the number of users
   D. Increasing the size of the SGA

12. The junior DBA decides to examine the Alert log each day to determine whether performance problems exist in the database. Which of the following types of information might the junior DBA find in the Alert log (choose three)?
   A. Instance startups and shutdowns
   B. Redo log switches
   C. Creation of database users
   D. Tablespace creation

13. What is the location in the SGA where Oracle keeps the most-recently executed SQL called?
   A. Large Pool
   B. Database Buffer Cache
   C. Redo Log Buffer
   D. Shared Pool

14. Which of the following commands would not cause sorting?
   A. ORDER BY
   B. ANALYZE TABLE
   C. SELECT DISTINCT
   D. INTERSECT
   E. All of the above may cause sorting to occur.

15. When creating a resource consumer group, resource plan, or resource plan directive, which of the following is true? (Choose all that apply.)
A. Each parameter must be preceded by a =>.
B. Each parameter must be enclosed in quotes.
C. The command must be typed all on one line.
D. All of the above are correct.

16. When a deadlock situation occurs, which session’s statement will be rolled back in order to resolve the deadlock?
A. The session that causes the deadlock
B. The session that detects the deadlock
C. The session that has done the least amount of work
D. The session that logged in last

17. A DBA has three goals for managing her databases: recoverability, reliability, and scalability. Which of these three goals will Oracle’s Shared Server feature address?
A. Performance
B. Scalability
C. Reliability
D. Security

18. You are interested in generating Explain Plans for a number of SQL statements that you suspect need tuning. You want to save the plan information for later evaluation. What is the first thing that you should do?
A. Run TKPROF.
B. Run the SQL Statement with the “Explain plan for…” syntax.
C. Set AUTOTRACE on in your SQL*Plus session.
D. Run utlxplan.sql.

19. You create a table with the following syntax:

```
CREATE TABLE EMP_HOLD
AS SELECT * FROM EMP NOLOGGING;
```

What should be done after this statement to ensure recoverability of the EMP_HOLD table?
A. Stop and Start the Oracle instance.
B. Issue a COMMIT statement.
C. Perform a hot or cold backup.
D. Perform a manual log switch.

20. All of the following are ways to gather hit ratio measurements on the database buffer cache except which?
   A. UTLBSTAT/UTLESTAT
   B. STATSPACK
   C. V$SYSSTAT
   D. V$DB_CACHE

21. An organization has decided to increase the number of users that will be interfacing with their Oracle application. What type of tuning life-cycle change is this?
   A. Design change
   B. Configuration change
   C. Workload change
   D. Application design change

22. Which of the following statements about the differences between the V$SESSTAT and V$SYSSTAT views are incorrect? (Choose two.)
   A. V$SESSTAT shows per session statistics, V$SYSSTAT shows instance wide statistics.
   B. V$SESSTAT includes the user name, V$SYSSTAT does not.
   C. V$SESSTAT shows statistics for connected sessions, V$SYSSTAT shows cumulative statistics for all sessions that have connected since instance startup.
   D. The statistics in V$SESSTAT and V$SYSSTAT are most accurate right after instance startup.
23. Oracle recommends keeping the CPU utilization below what percentage for optimal performance?
   
   A. 70  
   B. 80  
   C. 90  
   D. 100

24. If properly sized, the Shared Pool will cache application SQL and PL/SQL in memory 90 percent of the time or more in an OLTP system. This is important for which of the following reasons? (Choose three.)
   
   A. Reduce the overall memory overhead  
   B. Reduce I/O  
   C. Reduce CPU utilization  
   D. Minimize locking

25. Which of the following GUI tools is best suited to monitoring Oracle’s locking mechanisms?
   
   A. Top Sessions  
   B. Lock Monitor  
   C. Trace Data Viewer  
   D. SQL Analyze

26. Before undertaking a tuning effort on a database, benchmarks should be established. In which areas should these benchmarks be made? (Choose three.)
   
   A. Response time  
   B. Number of memory chips  
   C. CPU utilization  
   D. Number of server power supplies  
   E. Physical I/O and memory consumption
27. What observation can be made from the following query?

```
SQL> SELECT owner, segment_type, 
    2  segment_name, buffer_pool 
    3 FROM dba_segments;
```

<table>
<thead>
<tr>
<th>OWNER</th>
<th>SEGMENT_TYPE</th>
<th>SEGMENT_NAME</th>
<th>BUFFER_</th>
</tr>
</thead>
<tbody>
<tr>
<td>APPS</td>
<td>TABLE</td>
<td>EMPLOYEE</td>
<td>DEFAULT</td>
</tr>
<tr>
<td>APPS</td>
<td>TABLE</td>
<td>REGION</td>
<td>KEEP</td>
</tr>
<tr>
<td>APPS</td>
<td>TABLE</td>
<td>DIVISION</td>
<td>RECYCLE</td>
</tr>
<tr>
<td>APPS</td>
<td>INDEX</td>
<td>EMPLOYEE_FIRST_NAME_IDX</td>
<td>DEFAULT</td>
</tr>
<tr>
<td>APPS</td>
<td>INDEX</td>
<td>EMPLOYEE_ID_PK</td>
<td>RECYCLE</td>
</tr>
<tr>
<td>APPS</td>
<td>TABLE</td>
<td>SALES_HISTORY</td>
<td>RECYCLE</td>
</tr>
</tbody>
</table>

A. The REGION table is probably a very large, infrequently used table.
B. Blocks of the DIVISION table will be placed at the MRU end of the LRU list.
C. The REGION table is probably a small code table.
D. We are interested in preserving blocks of the SALES_HISTORY table in the database buffer cache.

28. Which one of these processes is responsible for copying data from the Online Redo Logs to the archive logs?

A. ARC0
B. LGWR
C. DBW0
D. PMON

29. You want to create a partitioned table based on a unique column. You want to ensure that Oracle will maintain a fairly even distribution of rows per partition. What type of partitioning solution would ensure a high degree of distribution among the given partitions?

A. Hash
B. Range
C. List
D. All of these partitioning types would work equally well.
30. A DBA is investigating a problem with a very large SQL statement and needs to see the entire SQL statement as it was executed. The best view to use to find the entire text of the SQL would be:

A. V$SQLAREA
B. V$SQLTEXT
C. V$SQL
D. V$SQL_PLAN

31. The DBA is interested in setting up tracing for a client that is running a third-party application. Since he does not have direct access to the application code or the client session, what is the proper method to use to initiate this tracing session?

A. Set SQL_TRACE=TRUE in the init.ora file.
B. Execute DBMS_SYSTEM.SET_SQL_TRACE_IN_SESSION for the session that he wants to trace.
C. ALTER SESSION SET SQL_TRACE=TRUE for the client session.
D. It is not possible to trace another client session directly.

32. A DBA is developing departmental guidelines for database performance tuning. These guidelines should include which of the following items? (Choose three.)

A. Performance tuning is an iterative process.
B. Performance tuning may be necessary when modifications are made to the system architecture.
C. Performance tuning plans can differ depending on the type of system being run.
D. Performance benchmarks should be established for the system before undertaking a tuning effort.
E. Performance tuning need only be performed once.

33. Which view can be queried to see the result of advice gathered on the database buffer cache?
A. V$DB_CACHE_ADVICE
B. V$SYSSTAT
C. V$SGASTAT
D. V$SGA

34. Which of the following would *not* be part of Oracle’s redo mechanisms?
   A. Log Writer
   B. Database Writer
   C. Checkpoints
   D. Archive Redo Logs

35. In order to improve the performance of an existing application, the DBA is considering adding additional CPUs to a database server. What types of application SQL will benefit the most from the presence of multiple CPUs?
   A. DML statement
   B. Parallel statement
   C. DDL statement
   D. Any PL/SQL packages and procedures

36. What is the special keyword used to define the high value for certain types of partitioned tables?
   A. UPPER
   B. MAXVALUE
   C. MAXIMUM
   D. UBOUND

37. Under which of the following conditions would the Java Pool most likely be used?
   A. When using an application utilizing JavaScript
   B. When using Java Applets
   C. When accessing an application that uses Java Stored Procedures
   D. When connecting with a JDBC connection
38. Oracle's default locking mechanism is at the row level. This level of locking helps guarantee a high degree of what?
   A. Data consistency
   B. Data concurrency
   C. Data accuracy
   D. Data extensibility

39. In what way do locally managed and dictionary managed tablespaces differ?
   A. Locally managed tablespaces use system tables to keep track of allocated and unallocated space, dictionary managed tablespaces do not.
   B. Locally managed tablespaces cannot be used to store application tables and indexes, dictionary managed tablespaces can.
   C. Locally managed tablespaces use bitmaps in their Datafiles to keep track of allocated and unallocated space, dictionary managed tablespaces do not.
   D. Locally managed tablespaces can only utilize the database's primary block size, dictionary managed tablespaces can have a different block size.

40. What is another name for the compiled version of SQL code that is cached in the Shared Pool?
   A. Dynamic link library
   B. P-Code
   C. script
   D. S-Code

41. Which optimization method considers many different execution plans and then selects the one with the lowest relative execution cost?
   A. Rule-Based Optimization
   B. Value-Based Optimization
   C. Cost-Based Optimization
   D. Performance-Based Optimization
42. Oracle recommends you use an OFA-compliant architecture when setting up your Oracle Server. What does OFA stand for?
   A. Oracle Flexible Architecture
   B. Oracle FailSafe Architecture
   C. Optimal Flexible Architecture
   D. Optimal FailSafe Architecture

43. The DBA wants to increase the size of the database buffer cache by 100MB. Currently, the overall size of the SGA is 256MB, the SGA_MAX_SIZE has been set to 300MB, and total system memory available to Oracle on the machine is 300MB. What should the DBA do?
   A. Issue ALTER SYSTEM commands to dynamically modify the SGA_MAX_SIZE and DB_CACHE_SIZE.
   B. Issue ALTER DATABASE commands to dynamically modify the SGA_MAX_SIZE and DB_CACHE_SIZE.
   C. Shut down, change the SGA_MAX_SIZE and DB_CACHE_SIZE parameters, and restart the instance.
   D. The DBA cannot increase the size of the database buffer cache by 100MB.

44. Which of the following would not be a characteristic of an SQL statement that may require further tuning?
   A. A statement that consumes excess CPU resources
   B. A statement that has a long parse, fetch or execute time
   C. A statement that reads too many data blocks from the SGA and too few from disk
   D. A statement that accesses many data blocks but returns only a few rows

45. You are setting up a new database. You have set up tempfiles to use for sorting and you would like to ensure that no users created in the database will use the system tablespace as their default temporary tablespace. What command could you issue to prevent this?
A. ALTER DATABASE DEFAULT TEMPORARY TABLESPACE temp;
B. ALTER TABLESPACE DEFAULT TEMPORARY TABLESPACE temp;
C. ALTER SYSTEM DEFAULT TEMPORARY TABLESPACE temp;
D. ALTER DATABASE DATAFILE temp DEFAULT TEMPORARY TABLESPACE temp;

46. The DBA has determined that Free List contention is occurring on several busy application tables. What is one method the DBA can use to minimize Free List contention on these tables?
   A. The FREELISTS keyword in the CREATE TABLE statement
   B. Automatic Segment-Space Management
   C. The FREELISTS init.ora parameter
   D. Drop and re-create the table with more Free Lists.

47. Which one of these statements is true about Oracle Shared Server connections?
   A. Many Shared Server processes can be serviced by a single client connection.
   B. Clients place their requests in a Request Queue.
   C. Dispatchers can service many clients simultaneously.
   D. There must be one Shared Server process for every Dispatcher process.

48. Generally, when interpreting the results of the Explain Plan, which operations in the plan are the ones that had been executed first in the SQL statement examined?
   A. The first operation listed
   B. The last operation listed
   C. The innermost operation when formatted as a nested tree
   D. The outermost operation when formatted as a nested tree
49. The minimum number of Redo Logs that an Oracle database must have is:
   A. One
   B. Two
   C. Four
   D. Eight

50. The Oracle ini.ora parameter that tracks the number of CPUs on a server is what?
   A. SERVER_CPUS
   B. CPU_COUNT
   C. CPU_NUMBER
   D. NUM_CPUS

51. If a transaction starts in an extent of a rollback segment, but the extent is not large enough to hold the entire contents of the transaction, what will occur?
   A. The transaction will fail to complete.
   B. The transaction will continue into the next available rollback segment.
   C. The transaction will grow into the next extent of the rollback segment even if the extent has active transactions.
   D. The transaction will wrap into the next rollback segment if it contains no active transactions.

52. What role needs to be granted to users that want to run AUTOTRACE in SQL*Plus?
   A. AUTOTRACE
   B. PLUSTRACE
   C. TRACING
   D. STATISTICS
53. The DBA has decided that some application queries will benefit from execution in parallel. What optional area of the SGA should be configured if Parallel Query is to be used?
   A. Redo Log Buffer
   B. Database Buffer Cache
   C. Shared Pool
   D. Large Pool

54. Which of the following is FALSE about AUTOTRACE (choose all that apply)?
   A. You need to belong to a specific role to use the tool.
   B. You need to SET AUTOTRACE ON to use the tool.
   C. You must have a PLAN_TABLE available to you.
   D. You can use the AUTOTRACE utility using third-party tools.

55. What statistic do you query for in V$SYSTEM_EVENT when examining the instance for Free List contention?
   A. free list
   B. buffer busy waits
   C. segment header
   D. None of the above

56. The three primary areas of server resources that affect system performance are what? (Choose three.)
   A. CPU
   B. Number of processes running on the server
   C. Disk I/O
   D. Memory

57. What internal Oracle structure keeps track of users who are waiting to obtain locks?
A. Latch
B. Buffer
C. Segment
D. Enqueue

58. If you want to see the last time that a table was analyzed, you would look in what DBA view?
A. DBA_SEGMENTS
B. DBA_TABLES
C. DBA_EXTENTS
D. DBA_OBJECTS

59. Which of the following methods of activating user tracing can be done by the DBA to another connected user?
A. ALTER SESSION SET SQL_TRACE=TRUE;
B. Use the DBMS_SYSTEM.SET_SQL_TRACE_IN_SESSION package.
C. Use the DBMS_UTILITY.SET_SQL_TRACE_IN_SESSION package.
D. Any of the above can be used to activate tracing.

60. Which of these steps would not take place during the parsing of an SQL statement?
A. Object names would be resolved.
B. Statement syntax would be checked.
C. The first set of rows is retrieved from the server.
D. A compiled version of the statement is generated.

61. Which of the following is made up of a collection of resource plan directives?
A. Resource consumer group
B. DBMSRESOURCE_MANAGER
C. Resource plan
D. Consumer plan group
62. Which of the following is TRUE about using the COMPUTE parameter of the ANALYZE TABLE command when generating statistics?

A. It should be used for very large tables.
B. It only uses a sampling of rows in the table to generate statistics.
C. It can be used with the SAMPLE clause.
D. It evaluates all of the rows in the table to generate statistics.

63. An update statement is issued on a row in an Oracle9i database. How long will the lock be held on the row?

A. Until a COMMIT or ROLLBACK is issued
B. The time duration is Operating System dependent.
C. 1 hour
D. It depends on the setting of the LOCK_TIMEOUT parameter in the init.ora file.
Answers to Assessment Test

1. A. These are the kernel parameters. These parameters are similar to the init.ora parameters for the Oracle Server and control the size and threshold values for certain structures in the operating system, such as the maximum number of processes that the operating system can be executing simultaneously. See Chapter 10 for more information.

2. D. Latches protect access to memory structures in an Oracle environment. This protection is necessary to ensure that only a single process is updating certain information in memory at any given moment. This ensures the integrity of the structures being accessed. See Chapter 9 for more information.

3. A. The EXPLAIN PLAN FOR syntax does not execute the query. Only the Explain Plan is created. This is an advantage over the AUTOTRACE facility, which will always execute the query. See Chapter 3 for more information.

4. B. The SYS and SYSTEM database users are members of the SYS_GROUP resource consumer group. See Chapter 10 for more information.

5. B. The EMPTY_BLOCKS column displays information on the total number of blocks above the High Water Mark. These are blocks that have never contained any data. See Chapter 8 for more details.

6. A. The default Oracle9i configuration uses Dedicated Server connections to the Oracle Server. Oracle clients are serviced by separate dedicated processes. See Chapter 6 for more details.

7. D. This is an example of composite partitioning, which is a combination of two different partitioning techniques. This is a common example of doing a primary partition by range and then doing subpartitions using hash partitioning. See Chapter 3 for more information.

8. C. Only one of the database’s resource plans can be defined at the instance level at a time. See Chapter 10 for more information.

9. D. When a log switch occurs to an Online Redo Log that has not been archived, the LGWR process will wait until archiver has completed archiving the log file before overwriting the information. This should not occur in a properly tuned Oracle database and is an indication that logs are too small or there are insufficient Online Redo Log groups. See Chapter 7 for more information.
10. C. Doing a SHUTDOWN ABORT does not cause a checkpoint to occur, so no dirty blocks will be written by DBW0. A shutdown of this type means that instance recovery is necessary when the database is restarted. Placing the database in hot backup mode, shutdown immediate, and forcing a log switch all cause DBW0 to write dirty blocks to disk. See Chapter 5 for more details.

11. C. Decreasing the number of users, while possibly effective, is not one of the two simple tenets of performance tuning. All the other options are examples of either “add more” or “make it bigger.” See Chapter 1 for more information.

12. A, B, and D. Examples of information contained in the Alert log include instance startups and shutdowns, Redo log switches, tablespace and datafile creation, block corruption errors, and segment space allocation errors. See Chapter 2 for more information.

13. D. The Shared Pool is the location in the SGA where Oracle holds the most-recently executed SQL and PL/SQL statements. The Database Buffer Cache is the location where Oracle retains the most-recently used data. The Redo Log Buffers contain recovery information for recently executed data modifications, and the large pool is used for, among other things, the Oracle Shared Server. See Chapter 4 for more details.

14. E. Any of these commands could cause sorting to occur. Sorting can also be caused by other commands such as doing SELECT statements with joined tables and using the UNION operator. See Chapter 8 for more details.

15. A, C. Not all parameters used when creating a resource consumer group, resource plan, or resource plan directive are in quotes. For example, the parameters for CPU emphasis and degree of parallelism are not enclosed in quotes. See Chapter 10 for more information.

16. B. The session that detects the deadlock will have its deadlocking statement rolled back in order to resolve the deadlock situation. See Chapter 9 for more information.

17. B. Scalability is the main issue addressed by Oracle Shared Servers. You may want to consider running in a Shared Server environment if your database server is reaching process limits or running tight on memory. See Chapter 6 for more details.
18. D. The first thing to do is run the utxplan.sql script. This script creates the PLAN_TABLE that is used to store execution plans. See Chapter 3 for more information.

19. C. Because the insertions of the data into the new table are not logged, the table would not be recoverable until it was backed up with either a hot or cold backup. See Chapter 7 for details.

20. D. V$DB_CACHE is not a valid Dynamic Performance View. All other methods of collection are valid. See Chapter 5 for more details.

21. C. Increasing the number of users that are utilizing the application would constitute a workload change. See Chapter 1 for more information.

22. B, D. The V$SESSTAT view shows statistics for all sessions that are connected to the instance. The V$SYSSTAT view does not contain any session-specific statistics, only summary statistics for the entire instance. Since the contents of the V$ views are cleared at instance shutdown, no V$ view contains useful statistics right after instance startup. See Chapter 2 for more information.

23. C. CPU utilizations above 90 percent will cause all processes running on the server to suffer poor performance. See Chapter 10 for more information.

24. A, B, and C. All of these are valid reasons for Oracle to cache SQL statements. By caching SQL, Oracle reduces the I/O overhead because it does not have to go to disk to retrieve information from the Data Dictionary. It reduces memory utilization because it does not have to allocate additional memory for identical statements and reduces CPU utilization because it does not have to re-parse the statements. See Chapter 4 for more details.

25. B. The Lock Monitor component of the Oracle Enterprise Manager Diagnostics Pack provides real-time information about the locks being taken by application users. See Chapter 2 for more information.

26. A, C, and E. All areas of the application and database should be monitored when establishing performance benchmarks. These areas include throughput and response times of SQL statements, statistical data on CPU and memory consumption, and information regarding physical I/O. See Chapter 1 for more information.
27. C. The blocks from the REGION table have been allocated to the Keep pool. Generally, these types of tables would be small, no larger than 10 percent of the overall database buffer cache size. See Chapter 5 for more details.

28. A. The ARC0 process would be responsible for moving information from the Online Redo Logs to the archive logs. You can have multiple Archive processes doing this work if one is not sufficient to keep up with the archive load. See Chapter 7 for details.

29. A. Hash partitioning would be the best solution when the goal is to maintain a fairly even distribution of rows among given partitions. Oracle uses a hashing algorithm to distribute the rows among the given partitions based on the partition key value. See Chapter 3 for more information.

30. B. The key to this question is that the DBA needs to see the ENTIRE SQL statement. V$SQLTEXT would provide the entire text of the SQL. V$SQL only contains the first 1000 characters of the SQL statement. V$SQLAREA also does not display the entire text of the SQL statement and V$SQL_PLAN contains the address of the SQL statement and not the actual text. Remember though, the SQL contained in V$SQLTEXT may be contains in several rows, so when you perform the query you would want to use an ORDER BY clause. See Chapter 4 for more details.

31. B. Using the DBMS_SYSTEM.SET_SQL_TRACE_IN_SESSION packaged procedure is the best method to use to trace another client processes session. You should not set SQL_TRACE to TRUE for the entire instance for this will generate trace files for all sessions all the time. The ALTER SESSION command will only trace the connected users session. See Chapter 2 for more information.

32. A, B, C, and D. All of these statements are true regarding performance tuning. See Chapter 1 for more information.

33. A. The V$DB_CACHE_ADVICE dynamic performance view contains statistics about performance of the database buffer cache at varying sizes. This is a tool that is designed to assist the DBA in achieving the optimal size of the database buffer cache. Remember to set the DB_CACHE_ADVICE init.ora parameter equal to ON to begin collection of these statistics. See Chapter 5 for more details.
34. B. Database Writer is not considered part of Oracle’s redo mechanisms. The Redo log buffer, Online Redo Logs, checkpoints, Archiver process, and archive logs constitute the components of Oracle redo architecture. See Chapter 7 for details.

35. B. Parallel statements can take advantage of multiple CPUs. The individual parallel processes can be divided among the available CPUs and run simultaneously to perform the overall statement more efficiently. See Chapter 10 for more information.

36. B. The upper boundary of a partitioned table should include a MAXVALUE partition so that no inserted rows fall above the given range of values for the column on which partitioning is being done. See Chapter 3 for more information.

37. C. The Java Pool would be utilized when you are using Server-Side Java. Java Stored Procedures would be an example of Server-Side Java. See Chapter 6 for more details.

38. B. Row-Level locking ensures a high degree of data concurrency. With row-level locking, it is much less likely that two users will affect one another by requesting the same lock on the same row at the same time. This locking does not prevent readers from accessing the data in the locked row and obtaining a read-consistent view of the data. See Chapter 9 for more information.

39. C. Dictionary managed tablespaces use data dictionary tables stored in the SYSTEM tablespace to determine whether space in the tablespace is free or allocated to a segment. Locally managed tablespaces use bitmaps stored in each Datafile’s header to determine whether blocks within the tablespace are free or in use. See Chapter 8 for more details.

40. B. P-Code is another name for the compiled version of code that Oracle creates during the parse phase of SQL execution. See Chapter 4 for more details.

41. C. The Cost-Based optimizer analyzes the statistics gathered on the objects in the SQL statement, considers many different execution plans, and then selects the one with the lowest relative execution cost. See Chapter 3 for more information.

42. C. The Optimal Flexible Architecture is a method of directory naming and location settings that enables you to have a common directory tree and naming construct for all critical Oracle software and application components. See Chapter 2 for more information.
43. D. Although increasing SGA_MAX_SIZE and DB_CACHE_SIZE and bouncing the instance would be the method to use if we had enough memory, in this case there is not enough memory available to make the change that we want. See Chapter 5 for more details.

44. C. All of these could indicate a statement needs tuning except reading blocks for the SGA. You would like to have more reads from the SGA and less reads from disk. This would lead to better performance. See Chapter 3 for more information.

45. A. You would use the ALTER DATABASE DEFAULT Temporary TABLESPACE temp; command to set the default temporary tablespace for users of the database. This should be done to avoid the use of the SYSTEM tablespace for sorting. See Chapter 8 for more details.

46. B. Automatic segment-space management utilizes bitmaps in the tablespace’s datafile headers, instead of Free Lists, to manage the free block allocations for each segment in that tablespace. This reduces the latch problem that can occur with Free Lists in the segment header. See Chapter 9 for more information.

47. C. Dispatchers can service many clients simultaneously. Because of this, you get a better utilization of resources. See Chapter 6 for more details.

48. C. Generally, the innermost operations of the execution plan are the ones that are the first executed. See Chapter 3 for more information.

49. B. You must have at least two Online Redo Logs—one that is the current Online Redo Log and one to switch to if the current one fills up. There is no limit to the number of online logs you can have. See Chapter 7 for details.

50. B. CPU_COUNT is the init.ora parameter that tells Oracle the number of CPUs available on the server. See Chapter 10 for more information.

51. D. Transactions must start and end in the same rollback segment. A transaction will attempt to wrap into the next extent of the rollback segment if the extent has no active transactions. If it does have active transactions, a new extent will be added to the rollback segment and the transaction will grow into this extent. See Chapter 8 for more details.

52. B. Grant the PLUSTRACE role to the users that are going to be using the AUTOTRACE utility in SQL*Plus. If they don’t belong to the role, they will receive an error if they try to set AUTOTRACE on. See Chapter 3 for more information.
53. D. One of the situations where the Large Pool can be used is when you are using parallel query commands. The Recovery Manager and Shared Server also utilize the Large Pool. See Chapter 6 for more details.

54. D. The AUTOTRACE utility is only available from SQL*Plus. Some third-party tools, such as TOAD, do provide features that are similar to AUTOTRACE. See Chapter 3 for more information.

55. B. Occurrences of the buffer busy waits statistic in V$SYSTEM_EVENT indicate that the instance may be experiencing Free List contention. See Chapter 9 for more information.

56. A, C, D. CPU, Memory, and Disk I/O are the three resources that need to be managed at the O/S level. These three components affect the performance of the server. See Chapter 10 for more information.

57. D. Enqueues are memory structures that keep track of users who are waiting to obtain locks. The enqueue keeps track of users who are waiting for locks, which lock mode they need, and in what order they asked for their locks. See Chapter 9 for more information.

58. B. The DBA_TABLES view will provide information regarding the last time that a table was analyzed. See Chapter 3 for more information.

59. B. A connected user’s session can be traced by the DBA by using the DBMS_SYSTEM.SET_SQL_TRACE_IN_SESSION PL/SQL package, or by setting the init.ora parameter SQL_TRACE=TRUE. See Chapter 2 for more information.

60. C. Rows would not be retrieved until the Fetch phase of SQL execution. During parsing, object names would be resolved, the statement would be compiled, and the statement syntax would be checked. See Chapter 4 for more details.

61. C. Resource plans are made up of one or more resource plan directives. The directives describe how database resources related to CPU and degree of parallelism should be allocated among users. See Chapter 10 for more information.

62. D. The COMPUTE parameter evaluates all of the rows in the table to generate statistics. For very large tables, it may be better to use the ESTIMATE parameter because of the overhead and time involved in doing a full COMPUTE. See Chapter 3 for more information.

63. A. Oracle has no timeout mechanisms on locks. Therefore, the lock would be held until a COMMIT or ROLLBACK command is issued. See Chapter 9 for more information.
Introduction to Performance Tuning

ORACLE9i PERFORMANCE TUNING EXAM OBJECTIVES COVERED IN THIS CHAPTER:

✓ Describe the roles associated with the database tuning process
✓ Describe the dependency between tuning in different development phases
✓ Describe Service Level Agreements
✓ Describe appropriate tuning goals
✓ Describe the most common tuning problems
✓ Describe the tuning considerations during development and production
✓ Describe performance and safety tradeoffs

Exam objectives are subject to change at any time without prior notice and at Oracle’s sole discretion. Please visit Oracle’s Certification website (http://www.oracle.com/education/certification/) for the current exam objectives listing.
Why is the system so slow?” The ability to answer this question definitively and authoritatively is one of the most important skills for an Oracle DBA to possess. Acquiring this skill requires a thorough understanding of the Oracle Server architecture, as well as knowledge of how Oracle processes SQL statements and how Oracle interacts with host operating systems and server hardware. Consequently, tuning an Oracle database can be a daunting task. Indeed, Oracle9i is so tunable and so feature-rich that it’s often difficult to know where to begin your tuning effort. Because of this, the application of proven tuning methodologies is the key to a successful tuning exercise.

This chapter will introduce the concepts related to selecting an appropriate performance tuning methodology and then discusses how to apply that methodology to produce a successful Oracle database tuning effort.

The Oracle Tuning Methodology

Oracle’s recommended tuning methodology has undergone some refinement in Oracle9i. Traditionally, Oracle recommended a top-down tuning approach that focused on application design and SQL tuning before examining tuning issues related to the database memory structures and physical I/O. While this methodology is still useful in some situations, Oracle now recommends that a more general set of “Performance Tuning Principles” be followed when the traditional top-down approach is not appropriate.

However, the two methodologies are not necessarily mutually exclusive. Instead, they are intended to help you recognize that the focus of your tuning efforts will be different in each situation. No matter which you choose,
performance tuning should be considered an on-going, iterative process—not one that ends when the first tuning issue is resolved. The two methodologies also recognize that the tuning effort must draw upon the expertise of not only the DBA, but also the application designers and developers, and system and network administrators.

The methodology you choose is generally dependent upon the type of system you are tuning. In general, the traditional approach is still well suited for tuning development databases, while the new, principle-based approach is generally more appropriate for tuning production systems.

### Tuning Development Systems

When tuning development systems, Oracle recommends a top-down approach that emphasizes examining the highest performance payoff areas first, and leaving areas with less impact on performance for later inspection. Table 1.1 shows the areas that this methodology focuses on and the order in which they should be examined.

#### Table 1.1 Oracle’s Top-Down Tuning Methodology

<table>
<thead>
<tr>
<th>Priority</th>
<th>Area to Examine and Tune</th>
</tr>
</thead>
<tbody>
<tr>
<td>First</td>
<td>Tuning the Data Design</td>
</tr>
<tr>
<td>Second</td>
<td>Tuning the Application Design</td>
</tr>
<tr>
<td>Third</td>
<td>Tuning Memory Allocation</td>
</tr>
<tr>
<td>Fourth</td>
<td>Tuning I/O and Physical Structure</td>
</tr>
<tr>
<td>Fifth</td>
<td>Tuning Resource Contention</td>
</tr>
<tr>
<td>Sixth</td>
<td>Tuning the Underlying Platform(s)</td>
</tr>
</tbody>
</table>
Using this methodology, application-related areas are examined before any database tuning issues are ever considered. This is an excellent strategy, since nearly 90 percent of all tuning issues are application-related and changes made to these first two areas can have a large impact on the remaining areas. For example, improvements to Application Design may necessitate changes in the areas of Memory Allocation and I/O. Therefore, the first two steps of the development tuning methodology focus on issues related to tuning the application design and SQL.

Once any application issues are addressed, the remaining four areas in this methodology focus on performance tuning topics that are directly related to the Oracle Server. These topics include tuning the memory structures, disk I/O activity, and resource contention in the database. Performance problems in these areas are identified using Oracle supplied scripts, queries against the data dictionary, and GUI tools such as Oracle Enterprise Manager, Oracle Expert, and Oracle Trace.

The Oracle9i Performance Tuning OCP Exam addresses only some of the topics related to the first two areas—Tuning the Data Design and Tuning the Application Design. These topics will be addressed in Chapter 3, “SQL Application Tuning and Design,” but only to the extent they are covered on the OCP exam. A full treatment of these topics is outside the scope of this book. The Oracle9i Performance Tuning Exam concentrates heavily on the last four areas—Tuning Memory, I/O and Physical Structure, Resource Contention, and Underlying Platforms. In general, the exam requires that you understand how each of these architecture mechanisms work, which measures are available to gauge the performance of these mechanisms, how to interpret the results of these measurements, and what changes are appropriate to improve the performance of these mechanisms. Additionally, topics such as the special tuning considerations for Decision Support Systems (DSS) and Online Transaction Processing Systems (OLTP) as well as the features of Oracle’s Shared Server option are also incorporated into the exam. These topics are discussed in Chapters 3 and 6, respectively.

**Oracle Objective**

Describe the tuning considerations during development and production.
Tuning Production Systems

Unlike development tuning, the "Performance Tuning Principles" that Oracle recommends for production systems focus on more general techniques to identify and resolve tuning issues. Oracle outlines these principles in Chapter 22, "Instance Tuning," of the *Database Performance Guide and Reference Release 1 (9.0.1)* documentation (Part Number A87503-02). They are also summarized in Table 1.2.

The full library of Oracle documentation is available online at Oracle's free Technet website: http://technet.oracle.com. This site also features trial software downloads and helpful user forums.

**TABLE 1.2** Oracle’s Production Performance Tuning Principles

<table>
<thead>
<tr>
<th>Priority</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>First</td>
<td>Define the problem clearly and then formulate a tuning goal.</td>
</tr>
<tr>
<td>Second</td>
<td>Examine the host system and gather Oracle statistics.</td>
</tr>
<tr>
<td>Third</td>
<td>Compare the identified problem to the common performance problems identified by Oracle in the <em>Oracle9i Database Performance Methods Release 1 (9.0.1)</em> documentation (Part Number A87504-02).</td>
</tr>
<tr>
<td>Fourth</td>
<td>Use the statistics gathered in the second step to get a conceptual picture of what might be happening on the system.</td>
</tr>
<tr>
<td>Fifth</td>
<td>Identify the changes to be made and then implement those changes.</td>
</tr>
<tr>
<td>Sixth</td>
<td>Determine whether the objectives identified in step one have been met. If they have, stop tuning. If not, repeat steps five and six until the tuning goal is met.</td>
</tr>
</tbody>
</table>
Notice that there are several areas of overlap between the traditional top-down methodology and this newer principle-based approach. One important difference is that the principle-based tuning approach recognizes the fact that changes to an application’s design are difficult, if not impossible, after a production system is in use. As packaged business applications become more prevalent, this tuning methodology becomes increasingly important since the highest priority tuning factors considered by the traditional top-down approach (data and application design) are almost entirely outside the control of the DBA when working with a packaged application.

Most of the objectives for the Oracle9i Performance Tuning OCP Exam are related to the second, fourth, and fifth principles (as listed in Table 1.2). They generally focus on examining the database for wait events—users or processes who are waiting for access to a database resource—and then trying to minimize these waits by making the appropriate changes to the system’s configuration.

Goal-Oriented Tuning

Whichever tuning methodology you choose, it is important to set specific tuning goals and then measure your progress toward those goals throughout the tuning process. This is not unlike the process a physician uses when treating a patient, where the physician records standard “performance” measures (i.e., blood pressure, temperature, respiration, heart rate, etc.) and then uses these measures throughout the course of treatment to track the progress of the patient’s condition. This process includes establishing benchmarks and gathering baseline statistics or timings against which you can measure your tuning progress, and setting measurable tuning goals.

Establishing a Benchmark

You should measure exactly how the system is currently performing before beginning any tuning effort. Using this benchmark, it is then possible to
formulate an idea of how you would like the system to perform after the initial tuning effort is complete. As in medicine, it is important to take a holistic view of the system when gathering this benchmark performance data. No single area should be examined in isolation. Some of the items that should be included in your initial measurements are:

- The performance of client machines used to access the system
- The performance of the network used to access the system
- The performance of the middle-tier application or web servers used to access the system
- The performance of the server hardware used to host the system
- The performance of the operating system used to manage the system hardware
- The performance of the Oracle database used as the repository for the application data

Ideally, all of these areas would be monitored simultaneously for a meaningful period of time, while the system is under a normal workload. This monitoring will likely require the expertise of many people, including system designers, application developers, system and network administrators, and power users. Outputs from monitoring consist primarily of statistics and timings.

**Statistics**

Important benchmark statistics can be gathered in all of the design, application, and developer areas listed above. These statistics might include:

- Client PC CPU and memory utilization
- Network traffic efficiency
- Server CPU and memory utilization
- Server disk I/O and controller activity
- Oracle instance memory utilization
- Oracle database I/O activity

These statistics can be gathered using vendor-supplied utilities, third-party tuning tools, or empirical observations. Oracle Server statistics are
typically gathered by running Oracle-supplied scripts, monitoring activity through the use of a GUI tool, or by examining database trace and log files. The resulting statistics are frequently expressed in terms of ratios, percentages, or wait times. Other statistics are expressed in terms of throughputs. *Throughput* is the amount of processing that a computer or system can perform in a given amount of time. An example of throughput might be, “How many customer deposits can we post to the appropriate accounts in four hours under regular workloads?” Throughput is an important measure when considering the scalability of the system. *Scalability* refers to the degree to which additional users can be added to the system without having system performance decline significantly. New features like Real Application Clusters make Oracle9i one of the most scalable database platforms on the market.

**Performance tuning considerations for Online Transaction Processing Systems usually revolve around throughput maximization.**

### Timings

Timings related to performance tuning are usually related to response times. *Response time* is the amount of time it takes for a single user’s request to return the desired result while using an application. An example of response time might be “How long does it take for the system to return a listing of all the customers who have purchased products that require service contracts?” Another example of tuning-related timings is time that users are spending waiting for access to a particular database resource.

**Performance tuning considerations for Decision Support Systems usually revolve around response time minimization.**

### Setting Performance Goals

Once you have established a good baseline against which you’ll measure your tuning activities, you need to establish measurable tuning goals. These goals should be specific and stated in a measurable way that allows you to use your existing data for comparison.
Tuning goals should be specific enough to allow you to quantitatively measure the benefits that your tuning efforts provide. For example:

- “I’d like to pin the five most frequently used PL/SQL packages in memory so that the users can access them faster.”
- “I’d like to reduce to 10 seconds or less the time it takes to enter a customer’s checking account deposit, post the funds to the customer’s account, and issue a receipt.”

These are both good examples of measurable tuning goals. They are much more useful tuning goals than something generic like, “I’d like the system to run faster.”

This explicit statement of goals is important for several reasons. First, it limits the tuning effort to a narrow scope. Changing too many factors at once has foiled many well-intentioned tuning campaigns. By limiting your tuning activities to one or two target areas, the effect of each change can be more easily measured and its effectiveness evaluated.

Second, explicit goals allow you to stop that part of the tuning process once the goals are achieved. While overall tuning is an ongoing and iterative process, it is possible to “tune a system to death” by endlessly tweaking obscure components that produce little or no real performance benefit. Setting tuning targets helps to prevent this problem by allowing you to move on to new tuning areas once you have achieved your specific goals. Specific areas where measurable tuning goals can be specified include the following:

- Reducing the frequency and duration of waits for database resources. These techniques are discussed in Chapter 9, “Tuning Contention.”
- Making sure that frequently accessed data and SQL statements are cached in memory. These techniques are discussed in Chapters 4, “Tuning the Shared Pool,” and Chapter 5, “Tuning the Database Buffer Cache.”
Minimizing response times and maximizing throughput. These concepts are discussed in Chapter 3, “SQL Application Tuning and Design.”

Service Level Agreements may also drive your performance tuning goals. Service Level Agreements, or SLAs, are written agreements between Database Administrators and their user communities. Some SLAs may specify that a particular report or process must complete within a specific period of time. SLAs may also include references to the maximum allowable downtime and recovery times for a system following media failure or user error. Each of these service level conditions represents areas that can be managed by proper performance tuning.

General Tuning Concepts

You now have an understanding of Oracle's tuning methodology and can appreciate the roles that goal setting and performance measurement play in this model. This section will build on these concepts by explaining how the consideration of tuning trade-offs, common problem areas, simple tuning guidelines, and the tuning lifecycle can enhance the performance tuning process.

Understanding Trade-offs

There is an old saying that states, “Oracle is not really a software company at all; it's merely a front for a consortium of CPU, disk, and memory manufacturers.” It probably seems that way to many purchasing managers after approving yet another purchase order for Oracle-related hardware.

However, this saying does demonstrate that, even with the application of a proven tuning methodology that utilizes extensive benchmarks, most tuning efforts involve some degree of compromise. This occurs because every Oracle Server is constrained by the availability of three key resources: CPU, disk (I/O), and memory.

CPU

Tuning Oracle’s memory and I/O activity will provide little benefit if the server’s processor is already overburdened. However, even in high-end,
multi-CPU servers, consideration should be given to the impact that changing memory or device configurations will have on CPU utilization. Oracle is also a very “CPU-aware” product. This means that several Oracle Server configuration parameters change dynamically when CPUs are added or removed from the server.

**Disk (I/O)**

The more Oracle Server activity that occurs in memory, the lower the physical I/O will be. However, placing too many demands on the available memory by oversizing Oracle’s memory structures can cause undesirable additional I/O in the form of operating system paging and swapping. Modern disk-caching hardware and software also complicate database I/O tuning, since I/O activity on these systems may result in reads from the disk controller’s cache and not directly from disk.

**Memory**

The availability of memory for Oracle’s memory structures is key to good performance. Managing that memory is important so that it is used to maximum benefit without wasting any that could be better used by other server processes. Oracle offers several memory tuning options that help you make the most efficient use of the available memory.

---

**NOTE**

Optional Oracle9i features designed to improve the database’s functionality in one area, may at the same time hinder its performance in others if those features are not configured properly. For example, Oracle9i offers several new high availability features that can maximize the database’s uptime and recoverability, but may also occasionally introduce additional performance tuning considerations in the areas of resource contention and server resource tuning.

**Common Tuning Problem Areas**

While examining your system for possible performance improvement, it is important to first examine those areas that are the most likely causes of poor
Performance. These areas include:

- Poorly written application SQL
- Inefficient SQL execution plans
- Improperly sized System Global Area memory structures
- Excessive file I/O
- Inordinate waits for access to database resources

**Oracle Objective**

Describe the most common tuning problems

**Poor Application SQL**

Inefficient application SQL is the cause of many database performance problems. In fact, inefficient SQL can cause all of the problems listed above. This occurs because inefficient SQL causes ineffective execution plans to be used when the statements are executed. Ineffective execution plans can cause excess I/O. Excess I/O can lower the effectiveness of the SGA’s memory structures, leading to excessive I/O and increased contention for other database resources. Tuning of application SQL is discussed in Chapter 3, “SQL Application Tuning and Design.”

**Inefficient Execution Plans**

Even well written application SQL can perform poorly if the Oracle optimizer does not formulate an effective execution plan for that SQL. The optimizer’s choice of execution plan is based largely on database statistics. Database tuning issues related to the Oracle optimizer are discussed in Chapter 3, “SQL Application Tuning and Design.”

**Improperly Sized SGA**

Oracle uses the SGA to cache application SQL and table data. The goal of this caching activity is to enhance performance by allowing application users to find requested SQL statements and data in memory whenever they request them. Therefore, sizing these structures incorrectly can cause poor database performance. Chapters 4, 5, 6, and 7 all address tuning the various components of the SGA.
Excessive File I/O

While many of the Oracle background processes must perform a certain amount of I/O during database operation, tuning these physical I/Os to avoid unnecessary disk reads and writes is key to optimal database performance. Tuning physical I/O is discussed in Chapter 8, “Tuning Disk I/O.”

Waits for Database Resources

Oracle uses a number of internal mechanisms to manage the access to the SGA’s memory structures and processes. When waits occur for access to these mechanisms, poor database performance can result. The monitoring and tuning of these mechanisms is discussed in Chapter 9, “Tuning Contention.”

Two Tuning Guidelines

Several different performance tuning topics and concepts will be discussed in subsequent chapters. Many of these topics involve new Oracle Server configuration parameters or the calculation of ratios and interpretation of data dictionary queries. When you focus on all these details, it’s easy to lose sight of the big tuning picture. Try to keep these two simple performance tuning tenets in mind while reading this book:

- Add more.
- Make it bigger.

At the end of a lengthy tuning analysis, either or both of these suggestions will frequently be the solution to your tuning problem.

Add More

Oracle’s performance tuning methodologies help identify which resources are causing bottlenecks because they are being demanded more frequently than they can be supplied. The easiest way to fix a performance problem of this type is to add more of that resource to the system.

Make It Bigger

Oracle’s performance tuning methodologies also help ascertain which resources are experiencing performance difficulties not because there are too few of that particular resource, but because the resource is too small to
service the requests made against it. The easiest way to fix a performance problem of this type is to make each of those resources larger.

The dreaded ORA-01555 “Snapshot Too Old” error, which occurs when a database’s rollback segments are too small and/or too few to service the database transactions properly, is a classic example of this concept. This error, and how Oracle9i’s System Managed Undo feature can be used to prevent it, is discussed in further detail in Chapter 8 of this book.

Tuning Lifecycles

The importance and relevance of database performance tuning experience peaks and valleys over the life of a system. While mature systems should always be monitored on an ongoing basis to determine if performance improvements can be achieved, there are generally three times when performance tuning is of even greater importance:

<table>
<thead>
<tr>
<th><strong>Oracle Objective</strong></th>
<th>Describe the dependency between tuning in different development phases</th>
</tr>
</thead>
</table>

**During application design**  As mentioned previously, the majority of application performance problems stem from poor design decisions made early in the development process. One example of this is excessively normalized database designs that require extensive joins to produce useful output.

**Configuring a new system**  Whenever new systems are implemented, important decisions regarding the choice of operating system, server hardware, optional Oracle features, and network connectivity will all have an impact on the performance of the system. Benchmark testing that accurately represents the workload on the final system is critical when comparing potential configurations.

**Changes in workload**  Existing systems that are currently performing acceptably may experience a drop in performance when the workload
on the system suddenly changes. Examples of this situation include adding a new module to an existing application suite or suddenly adding additional users to the system by making a portion of the application accessible to external customers via the Internet.

Review of the Oracle Architecture

To succeed as an Oracle DBA, you need to completely understand Oracle’s underlying architecture and its mechanisms. Understanding the relationship between Oracle’s memory structures, background processes, and I/O activities is critical before any tuning of these areas can be undertaken. The Oracle architecture model is reviewed here briefly.

The Oracle Server architecture can be broken into two main categories: logical (or memory) structures and physical (or file) structures. The Oracle Server memory structures are collectively known as the Oracle instance. Oracle’s physical file structures are collectively known as the Oracle database. These two structures taken together are called an Oracle Server. Both the traditional top-down, and newer principle-based performance tuning methodologies emphasize tuning memory structures before examining physical tuning of the database files.

The Oracle Instance

An Oracle Server instance is made up of Oracle’s main memory structure, called the System Global Area (SGA), and the Oracle background processes. The SGA, at a minimum, is made up of four components: the Shared Pool, the Database Buffer Cache, the Java Pool, and the Redo Log Buffer. Other components of the SGA, like the Large Pool, may also exist on your system depending on which optional Oracle9i features you choose to use. The Oracle background processes are, at a minimum: System Monitor (SMON), Process Monitor (PMON), Database Writer (DBW0), Log Writer (LGWR), and the Checkpoint Process (CKPT). Several other Oracle background processes may also be running on your system, like Recoverer (RECO), depending on which optional features you have decided to implement. Figure 1.1 summarizes the responsibilities of each of these mechanisms. Each of these areas will be discussed in greater detail with regard to performance tuning later in the book.
FIGURE 1.1 System Global Area Components

SGA Management
The components of the instance—the SGA and the required background processes—acquire space in the server's memory immediately upon startup of the database. While the required background processes will always acquire the same amount of memory for their purposes, four init.ora parameters are the primary determinants of SGA's memory requirements. These four parameters, along with their default values and descriptions, are shown in Table 1.3.
Although they are specified in the `init.ora`, several of these SGA-related parameters can also be dynamically modified after the instance is started. In previous versions of Oracle, the instance had to be shut down and then restarted before changes to SGA-related parameters would take effect. Because of this, adjusting these parameters in order to implement performance enhancements was difficult in high availability environments.
In Oracle9i the overall size of the SGA is defined by a single `init.ora` parameter: `SGA_MAX_SIZE`. Within this maximum, two of the individual components of the SGA—Shared Pool and Database Buffer Cache—can be dynamically resized using the `ALTER SYSTEM` command without shutting down and restarting the instance. If the value for `SGA_MAX_SIZE` is less that the sum of its components at instance startup, Oracle will ignore the setting for `SGA_MAX_SIZE` and use the sum of the components as the value instead.

As of this writing, neither the Large Pool nor the Java Pool can be dynamically resized using the `ALTER SYSTEM` command. However, Oracle has hinted that this may be an enhancement in subsequent releases of Oracle9i.

In order to achieve this dynamic SGA management, a new unit of SGA storage, the granule, was added to the Oracle architecture model. In Oracle9i, memory within the SGA is divided into units called granules. A granule is a piece of contiguous virtual memory allocated within the SGA. The size of each granule varies according to the overall size of the SGA as defined by the `init.ora` parameter `SGA_MAX_SIZE`. Table 1.4 shows the two possible sizes for SGA granules.

<table>
<thead>
<tr>
<th>Total SGA Size</th>
<th>Granule Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Less than 128MB</td>
<td>4MB</td>
</tr>
<tr>
<td>128MB or larger</td>
<td>16MB</td>
</tr>
</tbody>
</table>

When the instance is initially started, granules equal to the value specified by `SGA_MAX_SIZE` are created within the SGA. Each SGA component then acquires the space it needs by utilizing these available granules. Oracle enforces minimum granule requirements for each SGA component. These minimum values are shown in Table 1.5.

The minimum SGA size of an Oracle9i SGA is three granules, or 12MB.
If the SGA components do not utilize all the available granules, the excess granules will remain in the SGA, available for use when one of the SGA components is dynamically increased using the `ALTER SYSTEM` command. When dynamically increasing the size of an SGA component, there must be enough free granules available to accommodate the request, otherwise an ORA-02097 or ORA-04033 error is returned. Occasionally, you may have to reduce the size of one SGA component in order to increase the size of another, as Oracle does not perform this type of dynamic reallocation of space between SGA components automatically.

The use of granules as the unit of storage within the SGA also allows Oracle9i databases to contain tablespaces with differing Oracle block sizes. See Chapter 8, “Tuning Disk I/O,” for more information on multiple block sizes.

**User Process Management**

As shown in Figure 1.1, the Server Process allows the end user to interact with the SGA and its memory structures. In a Dedicated Server configuration each end user has their own Server Process. When the Shared Server option is used, several end-users share one or more Server Processes. User process and Server Process management can be an important aspect of overall database performance tuning. In particular, web applications that make repeated temporary connections to the database as a user using the application can contribute to performance degradation as the number of users increases over time.

<table>
<thead>
<tr>
<th>SGA Component</th>
<th>Minimum Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shared Pool</td>
<td>One granule</td>
</tr>
<tr>
<td>Database Buffer Cache</td>
<td>One granule</td>
</tr>
<tr>
<td>Redo Log Buffer</td>
<td>One granule</td>
</tr>
</tbody>
</table>

**TABLE 1.5** Minimum Sizes of SGA Components, In Granules
The Oracle Database

An Oracle database is made up physical files called control files, datafiles, and Redo logs. Additional physical files that are associated with the Oracle database are the init.ora, Trace and Alert log files, Password file, and any archived Redo log files. Table 1.6 summarizes the role that each of these files plays in the database architecture. Each of these areas will be discussed in greater detail with regard to performance tuning later in the book.

**TABLE 1.6** Oracle Physical Files

<table>
<thead>
<tr>
<th>File Name</th>
<th>Information Contained in File</th>
</tr>
</thead>
<tbody>
<tr>
<td>control file</td>
<td>Locations of other physical files, database name, database block size, database character set, and recovery information</td>
</tr>
<tr>
<td>Datafile</td>
<td>Data blocks from all database segments including tables and indexes</td>
</tr>
<tr>
<td>Redo log</td>
<td>A record of all changes made to the database, used for recovery</td>
</tr>
<tr>
<td>init.ora</td>
<td>A listing of configuration parameters for the instance and database</td>
</tr>
<tr>
<td>archived log file</td>
<td>Copy of the contents of previous online Redo logs, used for recovery</td>
</tr>
<tr>
<td>password file</td>
<td>Users who have been granted the SYSDBA and SYSOPER privilege</td>
</tr>
<tr>
<td>Alert log file</td>
<td>Informational messages and error messages generated by database activity</td>
</tr>
<tr>
<td>trace files</td>
<td>Informational messages and error messages generated by users or background processes</td>
</tr>
</tbody>
</table>

The init.ora file is of great importance to Oracle Server performance tuning. Many tuning modifications are implemented by making changes to the parameters in the init.ora. The default location for the init.ora is $ORACLE_HOME/dbs on Unix systems, and %ORACLE_HOME%\database on
Windows 2000 systems. On Unix systems the `init.ora` is generally a symbolic link to a file stored in `$ORACLE_BASE/admin/SID/pfile`, where `SID` is the name of the instance associated with that `init.ora`. Some `init.ora` parameters can be changed dynamically while the database is open, but others require that the instance be shut down and restarted before they take effect. For example, Oracle9i has the ability to dynamically change the size of the SGA’s components without having to shut down and start up the instance each time.

There are about 250 different documented `init.ora` parameters in Oracle9i. Thankfully, you do not need to know every one of them to succeed at performance tuning. The `init.ora` parameters related to performance tuning are discussed throughout this book.

Oracle9i also introduced the concept of Server Parameter (SP) files. Like the `init.ora`, these files are located in `$ORACLE_HOME/dbs` on Unix systems and `%ORACLE_HOME%/database` on Windows 2000 systems. The SP files are named `spfileSID.ora` on both operating systems. However, unlike traditional `init.ora` files, `spfile.ora` files will automatically incorporate any changes made dynamically via `ALTER SYSTEM` commands. An `init.ora` file will not reflect any dynamic system changes unless they are manually added to the `init.ora` file.

Unless the instance is specifically started using the `startup pfile` option with an existing `init.ora`, the values contained in the `spfile.ora`, not the `init.ora`, will be used to determine the instance parameters.

---

**Real World Scenario**

**Handle With Care: Undocumented `init.ora` Parameters**

You’ve just read a performance tuning tip posted to the Oracle newsgroup at `comp.databases.oracle.server`. The person posting the tip suggests setting the undocumented `init.ora` parameter `_dyn_sel_est_num_blocks` to a value of 200 in order to boost your database’s performance. Should you implement this suggestion?
Chapter 1 • Introduction to Performance Tuning

There are more than 400 undocumented init.ora parameters available in Oracle9i. Undocumented init.ora parameters are distinguished from their documented counterparts by the underscore that precedes their name, as with the parameter described in the newsgroup posting.

I do not recommend utilizing undocumented init.ora parameters on any of your systems because knowing the appropriate reasons to use these parameters, and the appropriate values to set these parameters to, is almost pure speculation because of their undocumented nature. While some undocumented parameters are relatively harmless, using others incorrectly can cause unforeseen database problems. What does the _dyn_sel_est_num_blocks parameter do and what value should you set it to? Only the engineers of the Oracle9i kernel code know for sure.

One exception to this suggestion is when you are directed to use an undocumented init.ora parameter by Oracle Worldwide Support. Oracle Support will occasionally use these parameters to enhance the generation of debug information or work around a bug in the kernel code. If you are asked to use an undocumented init.ora parameter by Oracle Support, you should include as a comment in the init.ora the Technical Assistance Request (TAR) number, date, and reason for using the parameter.

One init.ora parameter that is important to remember for database performance tuning is TIMED_STATISTICS. Whenever you undertake a tuning effort, you should set this parameter to TRUE and then shut down and restart the instance, or use the ALTER SYSTEM SET TIMED_STATISTICS=TRUE command to dynamically activate this feature. This parameter causes Oracle to gather timing information for actions taken against the database. The additional overhead inflicted on the system by these timings is far outweighed by the useful information they provide.

There are also many performance tuning modifications that can be made to the database without changing the init.ora. These modifications are made by issuing SQL commands against the Oracle Server, which are most often used to make structural changes to the database. Examples of these changes include modifying tablespaces, rollback segments, datafiles, and Redo logs. These commands are issued from the appropriate Oracle management tools like SQL*Plus or one of the Oracle Enterprise Manager tools.
Summary

Adopting a proven tuning methodology is the key to Oracle Server performance tuning. Since thorough tuning efforts require an analysis of the entire system, application designers, developers, and DBAs should all participate in the process. Oracle’s recommended tuning methodologies include a development-based top-down approach that stresses tuning application issues before physical database issues are addressed, and a principle-based approach of monitoring and then minimizing waits for various database resources. Taking into consideration all the trade-offs involved, tuning must take place within a framework of accurate benchmarks and measurable goals in order for the tuning effort to succeed.

Exam Essentials

Describe who is responsible for tuning. Know who, in addition to the DBA, should be involved in the database tuning effort.

Understand how lifecycle stages affect tuning. Understand how tuning requirements differ between the application design phase, the new system implementation phase, and workload change phase.

Describe the role of Service Level Agreements. Know how SLAs relate to the performance tuning process and how they affect tuning goals.

Understand the importance of goal setting. Be able to explain why setting performance goals is important and give examples of measurable tuning goals.

Know the performance problem areas. Describe what areas of the application and database make the best candidates for initial examination and tuning.
### Key Terms

Before you take the exam, be certain you are familiar with the following terms:

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alert log</td>
<td>Oracle Server</td>
</tr>
<tr>
<td>archived Redo log</td>
<td>Password file</td>
</tr>
<tr>
<td>Checkpoint Process (CKPT)</td>
<td>Process Monitor (PMON)</td>
</tr>
<tr>
<td>control files</td>
<td>Redo logs</td>
</tr>
<tr>
<td>database</td>
<td>Response time</td>
</tr>
<tr>
<td>Database Writer (DBW0)</td>
<td>scalability</td>
</tr>
<tr>
<td>datafiles</td>
<td>Service Level Agreements</td>
</tr>
<tr>
<td>granule</td>
<td>System Global Area (SGA)</td>
</tr>
<tr>
<td>init.ora</td>
<td>System Monitor (SMON)</td>
</tr>
<tr>
<td>instance</td>
<td>throughput</td>
</tr>
<tr>
<td>Log Writer (LGWR)</td>
<td>Trace</td>
</tr>
</tbody>
</table>
Review Questions

1. The Business group of a large organization is embarking on a new Information Technology project. Part of the requirements state that the system must be available 24 hours a day, Monday through Saturday, and that recovery processes should take no longer than one hour. Where are these types of requirements typically outlined?
   A. Systems Agreement
   B. Service Level Agreement
   C. Applications Charter
   D. DBA Contract

2. What is the first area to consider reviewing when doing a top-down tuning approach?
   A. Memory Allocation
   B. Resource Contention
   C. Data Design
   D. Underlying Platform(s)

3. A company has decided to deploy a large packaged financial system and use an Oracle database as the main repository for information. The DBA would typically use what approach to monitor system performance?
   A. Top-Down Approach
   B. Waterfall Tuning Approach
   C. Principle-based Production Tuning Approach
   D. Database Architecture Tuning Approach

4. Which of the following is not a tool that Oracle supplies to assist the DBA when addressing performance issues?
   A. UTLBSTAT/UTLESTAT
   B. Recovery Manager
   C. Oracle Enterprise Manager
   D. STATSPACK
5. Why is it important to set tuning goals?
   A. Goals allow you to focus your tuning efforts.
   B. Goals help identify system bottlenecks.
   C. Goals can be written and monitored via SQL scripts.
   D. None of the above is correct.

6. Benchmarks should be taken:
   A. For the database only
   B. For the network only
   C. For the client PCs only
   D. All of the above

7. Which of the following are not examples of a pretuning benchmark?
   A. Statistics
   B. Timings
   C. Both of these are good pretuning benchmarks.
   D. Neither of these is a good pretuning benchmark.

8. Which of the following is the best way to gather pretuning benchmarks on a system that utilizes PC clients that access the database over a wide-area network?
   A. Gather statistics and timings from the PC clients, network, and database, but not simultaneously.
   B. Gather the statistics and timings at a time when the system is not being utilized so as not to interfere with regular processing.
   C. Gather the statistics and timings for only a short period of time, preferably less than two minutes.
   D. Gather the statistics and timings for a useful period of time, during regular processing, on the PC clients, network, and database.
9. What does response time measure?
   A. The time it takes for a single user to receive a result setback when working from within the application.
   B. How much processing the system can do in a given amount of time.
   C. How fast a user’s PC can refresh the screen with new data.
   D. How long it takes for Oracle to find data without using an index.

10. What does throughput measure?
    A. The time it takes for a single user to receive a result set back when working from within the application.
    B. How much processing the system can do in a given amount of time.
    C. How fast a user’s PC can refresh the screen with new data.
    D. How long it takes for Oracle to find data without using an index.

11. Which of the following is not a way in which database performance statistics and timings are gathered?
    A. Using Oracle-supplied scripts.
    B. Using Oracle’s GUI tools.
    C. Examining Oracle log and trace files.
    D. All of these are good sources of database performance statistics.

12. Which of the following hardware resources is most impacted by performance tuning changes?
    A. The server CPU.
    B. The server memory.
    C. The server I/O system.
    D. All of the above are affected by tuning changes.
13. What is an Oracle instance made up of?
   A. SGA and the init.ora
   B. SGA and the background processes
   C. SGA, datafiles, control files, and Redo logs
   D. SGA, Alert log, and Trace files

14. An Oracle database is made up of:
   A. Datafiles, Redo logs, and trace files
   B. Datafiles, control files, and init.ora
   C. Datafiles, Redo logs, and control files
   D. Redo logs, archived Redo logs, and password file
   E. CKPT

15. The type of statistics gathered on the Oracle Server using vendor-specific tools may include which of the following?
   A. Network traffic efficiency
   B. Server disk I/O and controller activity
   C. Client PC CPU and memory utilization
   D. All of the above

16. A DBA determines that the current hit ratio for one of the SGA components is 85 percent. However, she feels that the ratio can probably be improved to 95 percent or better if additional resources are allocated to the SGA. This is a good example of which of the following?
   A. Maintaining good customer relations
   B. Goal-Oriented Tuning
   C. Efficient use of the DBA’s time and resources
   D. None of the above
17. Which of the following components of the System Global Area caches recently accessed data, index, and rollback blocks?
   A. Shared Pool
   B. Database Buffer Cache
   C. Large Pool
   D. Redo Log Buffer

18. The DBA notices that the system’s CPUs are constantly at 90 to 100 percent utilization. She decides to move some datafiles on disk to more evenly distribute the database I/O. What degree of benefit will this decision probably have on the overall system performance?
   A. A great deal of benefit
   B. Some benefit
   C. Very little benefit
   D. No benefit at all

19. An important parameter to set when the DBA is interested in gathering performance related statistics is:
   A. TIMED_STATISTICS
   B. PERFORMANCE_STATISTICS
   C. STATS_TIMING
   D. TIMED_STATS

20. The DBA increases the size of the SGA and causes the operating system to start swapping and paging additional processes from memory. What is the probable cause of this problem?
   A. Too few disk drives
   B. Too heavy a burden on the CPUs
   C. Too little memory
   D. Poorly written application code
Answers to Review Questions

1. B. Service Level Agreements outline these types of requirements and are an important component of most large IT projects.

2. C. Data Design is the correct place to start this type of tuning effort. Most performance problems are typically attributed to the Data Design and Application Design.

3. C. Principle-Based Production Tuning focuses on looking at system wait events and using techniques that are under the DBA’s control to minimize these waits. This is important, especially with package software, because generally the software vendor controls architectural changes.

4. B. All of these tools provide DBAs with tuning capabilities except Recovery Manager, which is used for backup and recovery of Oracle databases.

5. A. Tuning goals allow you to focus on areas of greatest importance first and then move on to lesser objectives after the goals are met.

6. D. All aspects of the system should have benchmarks established before testing the tuning process. This allows you to confirm that your tuning changes are having a positive effect and are not just coincidental to some other change.

7. C. Before any tuning changes are implemented, a good baseline measurement should be taken so that the effects of the tuning changes can be properly evaluated. System statistics and timings are both good examples of such benchmark statistics.

8. D. Accurate measures of pretuning performance must include all aspects of the system and should be taken over a long enough duration to provide meaningful results. These measurements should also be taken during a period of normal, or even heavy, processing so that the system will be under stress at the time.

9. A. Response time is an example of a measure that helps determine whether end users are benefiting from the tuning process.

10. B. Throughput is an example of a measure that helps determine whether batch processing and overall processing is benefiting from the tuning process.
11. D. All of these areas can provide useful statistics and timings. Each of these areas will be discussed in much greater detail later in the book.

12. D. CPU, memory, and I/O are all affected by tuning changes. For example, changes made to reduce I/O generally increase the demands on memory and CPU.

13. B. The System Global Area and its associated PMON, SMON, DBW0, LGWR, and CKPT background processes make up the Oracle instance.

14. C. An Oracle database consists of the control files, Redo logs, and datafiles. The trace files, archived Redo logs, password file, and init.ora are not considered part of the database.

15. B. Disk read/write activity and controller caching efficiency are two important server statistics that can be gathered using vendor-supplied tools.

16. B. Having a proper starting benchmark measurement and a target tuning objective are both key elements of a goal-oriented tuning technique.

17. B. The Database Buffer Cache holds the data, index, and rollback blocks that users are accessing via the application. Database Writer (DBW0) writes these blocks to the database datafiles.

18. C. Although moving files around to distribute I/O is generally a good exercise if your system is I/O bound, it will probably yield very little benefit if your system is CPU bound.

19. A. TIMED_STATISTICS is required to be set to TRUE if you want the Oracle Server to capture and maintain performance related statistics.

20. C. The probable cause is that the system is running short on memory. Two possible remedies would be to reduce the size of the SGA or increase the available memory on the Server.
Sources of Tuning Information

ORACLE9i PERFORMANCE TUNING EXAM OBJECTIVES COVERED IN THIS CHAPTER:

✓ Explain how the Alert log file is used
✓ Explain how background trace files are used
✓ Explain how user trace files are used
✓ Describe the statistics kept in the dynamic performance views
✓ Explain how STATSPACK collects statistics
✓ Collect statistics using STATSPACK
✓ Collect statistics using Enterprise Manager
✓ Use other tuning tools

Exam objectives are subject to change at any time without prior notice and at Oracle’s sole discretion. Please visit Oracle’s Certification website (http://www.oracle.com/education/certification/) for the current exam objectives listing.
As you saw in Chapter 1, “Introduction to Performance Tuning,” Oracle’s tuning methodologies emphasize the importance of measuring performance statistics throughout an Oracle Server tuning effort. These statistics are gathered through a variety of sources including system-generated logs and trace files, queries against dynamic performance views and data dictionary views, STATSPACK and UTLBSTAT/UTLESTAT reports, Oracle’s graphical management tools, and custom SQL scripts.

In addition to all of the following sources of performance tuning information, many Database Administrators also rely heavily upon custom SQL scripts to monitor important tuning statistics and events. Sources for these custom scripts include magazines, books, websites, and user groups. Many of these scripts can be found by doing a quick Internet search on the key words “Oracle tuning scripts.” This book also includes several examples of such scripts in subsequent chapters.

The Alert Log

The Oracle Alert log records informational and error messages for a variety of activities that have occurred against the database during its operation. These activities are recorded in chronological order from the oldest to most recent. The Alert log is found in the directory specified by the init.ora parameter BACKGROUND_DUMP_DEST. On systems that follow the Optimal Flexible Architecture (OFA) model recommended by Oracle, this directory will be $ORACLE_BASE/admin/SID/bdump on Unix systems and %ORACLE_BASE%/admin\SID\bdump on Windows 2000 systems, where SID represents the name of the Oracle instance that will be creating the Alert log.
The Alert Log

The Optimal Flexible Architecture model is discussed in detail in Chapter 1, “Introduction,” in the Oracle9i Database Migration, Release 1 (9.0.1) (Part Number A90191-02).

**Oracle Objective**

Explain how the Alert log file is used

The name of the Alert log varies by operating system. On Unix systems, it follows the naming convention of alert_SID.log. On Windows 2000 systems, it follows the naming convention of SIDArlt.log. In both cases, SID represents the name of the Oracle instance the Alert log belongs to. A sample Alert log from a Windows 2000 Oracle9i Server is shown in Figure 2.1.

**FIGURE 2.1** Sample section of Alert log file

```
Recovery of Online Redo Log: Thread 1 Group 3 Seq 591 Reading mem 0
Mon Jan 14 16:39:50 2002
Mon Jan 14 16:39:50 2002
End of recovery

MONK: enabling cache recovery
Mon Jan 14 16:48:54 2002
Undo Segment 11 Online
Undo Segment 12 Online
Undo Segment 10 Online
Undo Segment 14 Online
Undo Segment 15 Online
Undo Segment 16 Online
Undo Segment 17 Online
Undo Segment 18 Online
Undo Segment 19 Online
Undo Segment 20 Online
Successfully online Undo Tablespace IF.

SNCK: enabling redo recovery
Thread 1 advanced to log sequence 599
Current log# 3 seg# 599 mem# 0
DI: \ORACLE\ORADATA\PROD\RED003, RDO
MONK: enabling redo recovery
Thread 1 advanced to log sequence 599
Current log# 1 seg# 599 mem# 0
DI: \ORACLE\ORADATA\PROD\RED003, RDO

Errors in file B:\oracle\admin\PROD\dump\proddb.err
ORA-01584: attempt to write into rollback segment [18] extent [2] which is being freed
Thread 1 advanced to log sequence 599
Current log# 2 seg# 599 mem# 0
DI: \ORACLE\ORADATA\PROD\RED003, RDO

Mon Jan 14 16:58:33 2002
Errors in file B:\oracle\admin\PROD\dump\proddb.err:
ORA-01538: internal error code, arguments [1100], [0x37F377E5], [0x37F377E5], [1100].
```
The Alert log will frequently give you a quick indication of whether gross tuning problems exist in the database. Tables that are unable to acquire additional storage, sorts that are failing, and problems with rollback segments are all examples of tuning problems that can show up as messages in the Alert log. Because of this, you should examine your Alert log daily to see if any of these obvious tuning problems exist. Some common Alert log entries that are related to performance tuning issues are shown in Table 2.1. This is not a complete list, and not all of these messages will appear in every Alert log. Some of them can be found in the sample Alert log shown in the previous section.

**Table 2.1** Common Alert Log Messages Related to Performance Tuning

<table>
<thead>
<tr>
<th>Alert Log Message</th>
<th>Message Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unable to Extend Temp Segment by $n$ in Tablespace $x$</td>
<td>Identified by the error code ORA-01652, this error occurs when a user’s Server Process cannot find enough contiguous free space in their temporary tablespace to perform the required sort operation. Tuning of sort activity is discussed in Chapter 8, “Tuning Disk I/O.”</td>
</tr>
<tr>
<td>Unable to Extend Table $x$ by $n$ in Tablespace $x$</td>
<td>Identified by the error code ORA-01653, this error occurs when a table segment is attempting to acquire an additional extent, but cannot do so because there is not enough contiguous free space in the tablespace. Extent management is discussed in Chapter 8, “Tuning Disk I/O.”</td>
</tr>
<tr>
<td>Unable to Extend Rollback Segment $x$ by $n$ in Tablespace $x$</td>
<td>Identified by the error code ORA-01650, this error occurs when a rollback segment needs to grow, but is unable to because there is insufficient contiguous free space in the tablespace where the rollback segment resides. Rollback segment tuning is discussed in Chapter 8, “Tuning Disk I/O.”</td>
</tr>
<tr>
<td>Max # Extents $n$ Reached in Table $x$</td>
<td>Identified by the error code ORA-01631, this error occurs when a table segment needs to grow, but is unable to because it would exceed its maximum allowed number of extents. Segment extent tuning is discussed in Chapter 8, “Tuning Disk I/O.”</td>
</tr>
</tbody>
</table>
The Alert log also contains information related to backup and recovery issues, including datafile and redo log corruption. In particular, the Alert log should be examined regularly for ORA-00600 errors. These ambiguous internal errors are largely undocumented. However, they can be indicators of data corruption or other structural database problems. Resolution of these errors should be undertaken with the assistance of Oracle Worldwide Support at http://metalink.oracle.com.

Explanations of all the Oracle errors that are identified by ORA-codes can be found in the Oracle9i Database Error Messages, Release 1 (9.0.1) (Part Number A90202-02).

The Alert log is constantly appended to while the database is in operation. If left unmanaged, it can grow to be a very large file. While the information in the Alert log is an excellent tuning aid, a large Alert log containing information dating back several months or years is difficult to work with.
You can manage the Alert log’s size by regularly renaming, trimming, or deleting it. You may want to perform this maintenance following your nightly or weekly database backups. Oracle automatically creates a new Alert log if the prior one was renamed or deleted.

### Trimming the Alert Log

The Unix `tail` command can be used to manage the size of the Alert log. For example, a Unix shell script that contains the following commands would trim all but the last 1,000 lines from the Alert log for a database called PROD:

```bash
cd $ORACLE_BASE/admin/PROD/bdump
tail -1000 alert_PROD.log > alert_PROD.log.trimmed
mv alert_PROD.log.trimmed alert_PROD.log
```

### Background, Event, and User Trace Files

Oracle trace files are text files that contain session information for the process that created them. Trace files can be generated by the Oracle background processes, through the use of `init.ora` trace events, or by user Server Processes. Many of these trace files contain useful information for performance tuning and system troubleshooting. We will look at background, event, and trace files in the following sections.

---

**Oracle Objectives**

- Explain how background trace files are used
- Explain how user trace files are used

---

### Background Process Trace Files

Background process trace files are found in the directory specified by the `init.ora` parameter `BACKGROUND_DUMP_DEST`. On systems that follow the OFA model, this directory will be `$ORACLE_BASE/admin/SID/bdump` on
Unix systems and `%ORACLE_BASE%\admin\SID\bdump` on Windows 2000 systems, where `SID` is the name of the Oracle instance that generated the file. Background process trace files incorporate the name of the background process that generated them in the trace file name. Examples of Unix and Windows 2000 background trace file names are shown in Table 2.2.

**Table 2.2** Sample Background Process Trace File Names

<table>
<thead>
<tr>
<th>Process Name</th>
<th>Unix System</th>
<th>Windows 2000 System</th>
</tr>
</thead>
<tbody>
<tr>
<td>Process Monitor (PMON)</td>
<td>Pmon_nnnn.trc</td>
<td>sidPMON.trc</td>
</tr>
<tr>
<td>System Monitor (SMON)</td>
<td>Smon_nnnn.trc</td>
<td>sidSMON.trc</td>
</tr>
<tr>
<td>Database Writer (DBW0)</td>
<td>Dbw0_nnnn.trc</td>
<td>sidDBW0.trc</td>
</tr>
<tr>
<td>Log Writer (LGWR)</td>
<td>Lgwr_nnnn.trc</td>
<td>sidLGWR.trc</td>
</tr>
<tr>
<td>Checkpoint Process (CKPT)</td>
<td>Ckpt_nnnn.trc</td>
<td>sidCKPT.trc</td>
</tr>
<tr>
<td>Archive Process (ARC0)</td>
<td>Arc0_nnnn.trc</td>
<td>sidARC0.trc</td>
</tr>
</tbody>
</table>

**Event Trace Files**

The Oracle9i Server also has special options that you can configure to enable the collection of detailed tracing information for particular database events. A *database event* is a specific action or activity that occurs in the database. By default, most database events do not cause the creation of trace files. However, some database events can be forced to create a trace file of their activities by adding a tracing event parameter to the `init.ora` of the database you wish to trace. The trace files will be created in the location specified by the `BACKGROUND_DUMP_DEST` parameter in the `init.ora` file.

As part of a troubleshooting effort, you may occasionally be asked to activate tracing for database events by Oracle Worldwide Support as part of a troubleshooting or tuning effort.

Tracing of these events is usually done at the instance level by adding new lines to the `init.ora`. If several trace events are being added to the
The `init.ora` entries will look like this:

```
EVENT="10046 trace name context forever, level 12"
```

The double quotes are part of the actual syntax of this parameter. Notice that each event entry is made up of two components:

- The event to be traced, designated by a number (10046 in the example above)
- The level to which tracing will be performed, designated by a number (12 in the example above)

The numeric values for both the events and their associated levels are not widely documented. In most cases, Oracle Worldwide Support personnel will provide you with these values when the need for using trace events arises. Setting them to the proper values is important for the event tracing to work properly.

### User Trace Files

User trace files are found in the directory specified by the `init.ora` parameter `USER_DUMP_DEST`. On systems that follow the OFA model, this directory will be `$ORACLE_BASE/admin/SID/udump` on Unix systems and `%ORACLE_BASE%\admin\SID\udump` on Windows 2000 systems, where `SID` is the name of the Oracle instance that generated the file. User process trace files incorporate the OS process identifier in the trace file name. On Unix systems, they also incorporate the name of the instance that the user was connected to when they were generated. Examples of Unix and Windows 2000 background trace file names are shown here:

- Unix System: `ora_prod_4327.trc`
- Windows 2000 System: `ora00117.trc`

A query can be performed against the `V$PROCESS` and `V$SESSION` dynamic performance views in order to identify which user generated a particular trace file. The following code demonstrates how to match a user trace file to the user session that generated it.

```
SQL> SELECT s.username, p.spid
      2  FROM v$session s, v$process p
```
3  WHERE s.paddr = p.addr
4  AND p.background is null;

<table>
<thead>
<tr>
<th>USERNAME</th>
<th>SPID</th>
</tr>
</thead>
<tbody>
<tr>
<td>MATT</td>
<td>4292</td>
</tr>
<tr>
<td>DBSNMP</td>
<td>4377</td>
</tr>
<tr>
<td>SYSTEM</td>
<td>4372</td>
</tr>
<tr>
<td>JOE</td>
<td>4437</td>
</tr>
<tr>
<td>BRENDA</td>
<td>4368</td>
</tr>
<tr>
<td>EMILY</td>
<td>4469</td>
</tr>
</tbody>
</table>

Using the output from this SQL query, it can be seen that a trace file with the name `ora_prod_4292.trc` or `ora04292.trc` would belong to Matt’s session. Note that tracings from sessions that are connected via Shared Server processes, Transaction Managers that use the XA X/Open Interface, or the Oracle Call Interface (OCI) may generate more than one trace file per user session.

The following sections explain how to activate user tracing for the entire instance and individual users. Techniques for formatting the resulting trace files using the `TKPROF` utility can be found in Chapter 3, “SQL Application Tuning and Design.”

### Activating User Tracing

Oracle occasionally creates a user trace file when processing errors occur in a user’s Server Process. The code listing below shows a portion of the output from a user trace file that was generated on a Windows 2000 Oracle9i Server when a deadlock error occurred. Deadlock errors are discussed in more detail in Chapter 9, “Tuning Contention.”

```
Dump file d:\oracle\admin\SEED\udump\ORA00476.TRC
Fri Jan 04 20:57:08 2002
ORACLE V9.0.1.1.1 - Production vsnsta=0
vsnsql=10 vsnxtr=3
Windows 2000 Version 5.0 Service Pack 1, CPU type 586
Oracle9i Enterprise Edition Release 9.0.1.1.1 - Production
With the Partitioning option
JServer Release 9.0.1.1.1 - Production
Windows 2000 Version 5.0 Service Pack 1, CPU type 586
Instance name: prod
```
Redo thread mounted by this instance: 1

Oracle process number: 9

Windows thread id: 476, image: ORACLE.EXE

*** 2002-01-04 20:57:08.000
*** SESSION ID:(8.5) 2002-01-04 20:57:08.000
DEADLOCK DETECTED
Current SQL statement for this session:
update payroll.deductions set FICA = '120' where emp_id = '101'

The following deadlock is not an ORACLE error. It is a deadlock due to user error in the design of an application or from issuing incorrect ad-hoc SQL. The following information may aid in determining the deadlock:
Deadlock graph:

While events such as deadlocks automatically generate trace files, full scale tracing of user sessions does not occur unless the user or DBA requests it. Full user tracing can be activated at either the instance level or session level using any of the following methods.

**Instance-Level Tracing**
By setting the `init.ora` parameter `SQL_TRACE=TRUE`, all processes against the instance will create their own trace files. This particular method of tracing should be used with care since it creates a great deal of overhead against the system. The default value for this parameter is `FALSE`.

**User-Level Self Tracing**
Users can turn SQL tracing on or off in their own session using the following SQL commands:

```
ALTER SESSION SET SQL_TRACE=TRUE;
```
To start the tracing process, and

```
ALTER SESSION SET SQL_TRACE=FALSE;
```
to end the tracing process.
Any SQL statements issued between the time when `SQL_TRACE` is set to `TRUE`, and the time it is set to `FALSE`, will be recorded in the user’s trace file.
User-Level DBA Tracing

Most applications do not provide the user with a mechanism for issuing SQL commands from within the application. Therefore, the ALTER SESSION SQL commands shown in the previous section cannot be used to initiate tracing in these types of environments. In this situation you can initiate tracing against the user’s session using the Oracle-supplied PL/SQL package called DBMS_SYSTEM. This PL/SQL package contains a procedure called SET_SQL_TRACE_IN_SESSION that allows you to activate tracing in another user's session by supplying the system identifier (sid) and serial number (serial#) values for that user as shown in the V$SESSION view. The following example demonstrates how first to identify, and then activate tracing in a session for the user Bob.

1. Identify the sid and serial# for Bob’s session by querying V$SESSION:
   SQL> select username, sid, serial#
       2   from v$session
       3   where username = 'BOB';

<table>
<thead>
<tr>
<th>USERNAME</th>
<th>SID</th>
<th>SERIAL#</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOB</td>
<td>10</td>
<td>2642</td>
</tr>
</tbody>
</table>

2. Activate tracing in Bob’s session by using the DBMS_SYSTEM PL/SQL package and the values for sid and serial#:
   SQL> exec sys.dbms_system.set_sql_trace_in_session(10,2642,TRUE);

3. Bob’s session is now generating a trace file listing to the directory specified in USER_DUMP_DEST in the init.ora. Once the activity you wish to trace is complete, you can disable the tracing on Bob’s session using the DBMS_SYSTEM package again.
   SQL> exec sys.dbms_system.set_sql_trace_in_session(10,2642,FALSE);

   Tracing is now stopped in Bob’s session.

Interpreting User Trace File Output

Once the user trace file is generated using any of the above methods, it can be used to aid many aspects of performance tuning. Details of how to
interpret the contents of user trace files using the TKPROF utility are covered in Chapter 3, “SQL Application Tuning and Design.”

Managing Trace Files

Once tracing is activated in a user’s session, every action the user takes against the instance is included in the trace file output. If tracing is left on for an extended period of time, the files generated may be very large. In fact, a trace file could grow large enough to fill the entire disk location specified in the USER_DUMP_DEST init.ora parameter.

The init.ora parameter MAX_DUMP_FILE_SIZE allows you to limit the size to which a user trace file can grow. This value can be expressed in either operating system blocks or bytes. Table 2.3 shows the various ways in which this parameter can be specified.

<table>
<thead>
<tr>
<th>Parameter Specification</th>
<th>Resulting Maximum Size of User Trace</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAX_DUMP_FILE_SIZE=10000</td>
<td>10000 OS blocks</td>
</tr>
<tr>
<td>MAX_DUMP_FILE_SIZE=500K</td>
<td>500,000 bytes</td>
</tr>
<tr>
<td>MAX_DUMP_FILE_SIZE=10M</td>
<td>10 megabytes</td>
</tr>
<tr>
<td>MAX_DUMP_FILE_SIZE=unlimited</td>
<td>No limit on file size</td>
</tr>
</tbody>
</table>

Like the Alert log file, background, event, and user trace files should also be renamed, trimmed, or deleted occasionally so it is easier to find the most recent trace file data when needed.

Performance Tuning Views

There are two sources of broad tuning information in Oracle’s internal views: the V$ dynamic performance views and the DBA data dictionary views.
The data contained in these two families of views are beneficial for both performance tuning benchmarking and subsequent performance monitoring. Before examining the contents of these views in detail, let’s look at a brief summary of how these two types of views differ:

- The names of the V$ views are generally singular whereas the DBA views usually have plural names. For example V$DATAFILE vs. DBA_DATA_FILES.

- Many of the V$ views are available when the database is in a nomount or mount state. DBA views are only available when the database is open.

- The data contained in the V$ views is generally lowercase. The data contained in the DBA views is usually uppercase. This is important to remember when specifying WHERE clauses in your queries against these views.

- The V$ views contain dynamic performance data which is lost each time the instance is shut down. Because of this, these views always contain statistics that have been gathered since the instance was started. From a performance tuning perspective, when you examine the data in these views can be as important as how you examine the data in these views. If the instance is up most of the time and rarely shut down, these statistics may encompass so much time that they are useless. The data contained in the DBA views is more static and is not cleared at instance shutdown. Because of the dynamic nature of the V$ views, you will find that they are the more commonly used data dictionary views for performance tuning.

The dynamic performance views and data dictionary views are discussed in greater detail in the following sections.

**V$ Views Commonly Used in Performance Tuning**

There are approximately 225 V$ dynamic performance views in an Oracle9i database. These views are based on dynamic tables collectively known as the X$ tables. These virtual tables exist only in memory and are largely undocumented, having cryptic names like X$KSMS and X$KWQSI. Therefore, most performance tuning information is obtained from the V$ views themselves. Table 2.4 contains a partial listing of the V$ views that are frequently used in performance tuning.
**Oracle Objective**

Describe the statistics kept in the dynamic performance views

### Table 2.4 Sample of Dynamic Performance Views

<table>
<thead>
<tr>
<th>View Name</th>
<th>View Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>V$SGASTAT</td>
<td>Shows information about the size of the System Global Area’s components.</td>
</tr>
<tr>
<td>V$EVENT_NAME</td>
<td>Shows database events that may require waits when requested by the system or by an individual session. There are approximately 200 possible wait events. Wait events are discussed in Chapter 9, “Tuning Contention.”</td>
</tr>
<tr>
<td>V$SYSTEM_EVENT</td>
<td>Shows events for which waits have occurred for all sessions accessing the system.</td>
</tr>
<tr>
<td>V$SESSION_EVENT</td>
<td>Shows events for which waits have occurred, individually identified by session.</td>
</tr>
<tr>
<td>V$SESSION_WAIT</td>
<td>Shows events for which waits are presently occurring, individually identified by session.</td>
</tr>
<tr>
<td>V$STATNAME</td>
<td>Matches the name to the statistics listed only by number in V$SESSTAT and V$SYSSTAT.</td>
</tr>
<tr>
<td>V$SYSSTAT</td>
<td>Shows overall system statistics for all sessions, both currently and previously connected.</td>
</tr>
<tr>
<td>V$SESSTAT</td>
<td>Shows statistics on a per-session basis for currently connected sessions.</td>
</tr>
<tr>
<td>V$SESSION</td>
<td>Shows current connection information on a per-session basis.</td>
</tr>
<tr>
<td>V$WAITSTAT</td>
<td>Shows statistics related to block contention.</td>
</tr>
</tbody>
</table>
In general, queries that incorporate V$SYSSTAT will show statistics for the entire instance since the time it was started. By joining this view to the other relevant views, you get the overall picture of performance in the database. Alternatively, queries that incorporate V$SESSTAT will show statistics for a particular session. These queries are better suited for examining the performance of an individual operation or process.

A complete description of all the V$ views can be found in Chapter 3, “Dynamic Performance (V$) Views,” in the Oracle9i Database Reference Release 1 (9.0.1) (Part Number A90190-01).

**DBA Views Commonly Used in Performance Tuning**

There are approximately 170 DBA data dictionary views in an Oracle9i database. The tables underlying these views are the physical tables that make up the Oracle data dictionary. Like the X$ virtual tables underlying the V$ views, these tables are also generally undocumented and have only mildly meaningful names like OBJ$ and FILE$. However, in most cases, the DBA views themselves will provide the information you need to begin performance tuning. Table 2.5 contains a partial listing of some of the DBA views that are used when you undertake performance tuning on a database.

**TABLE 2.5** Sample of Static Dictionary Views

<table>
<thead>
<tr>
<th>View Name</th>
<th>View Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBA_TABLES</td>
<td>Table storage, row, and block information</td>
</tr>
<tr>
<td>DBA_INDEXES</td>
<td>Index storage, row, and block information</td>
</tr>
<tr>
<td>INDEX_STATS</td>
<td>Index depth and dispersion information</td>
</tr>
<tr>
<td>DBA_DATA_FILES</td>
<td>Datafile location, naming, and size information</td>
</tr>
<tr>
<td>DBA_SEGMENTS</td>
<td>General information about any space consuming segment in the database</td>
</tr>
<tr>
<td>DBA_HISTOGRAMS</td>
<td>Table and index histogram definition information</td>
</tr>
</tbody>
</table>
A complete description of all the DBA views can be found in Chapter 2, “Static Data Dictionary Views,” in the Oracle9i Database Reference Release 1 (9.0.1) (Part Number A90190-01).

Some of the columns in the DBA views are not populated with data until the table or index the view references has been analyzed using the ANALYZE TABLE or ANALYZE INDEX commands.

Analyzing tables and indexes is discussed in Chapter 3, “SQL Application Tuning and Design.”

Sample Queries on Views

In order to demonstrate the usefulness of these views for performance tuning, a brief example of each type will be presented here. Each subsequent chapter also contains examples of relevant queries on the V$ and DBA views for the topic addressed in that chapter.

Sample V$ View Query

When querying the V$ views, it is not uncommon to join two or more of these views together in order to get all the information you require. For example, querying V$SESSTAT will give you detailed statistics about events within a particular user’s session, but will not tell you the name of each statistic. To get the meaning of each statistic, you will have to join V$SESSTAT to V$STATNAME. An example of this type of query is shown in the following code listing. This query would show all the session statistics for a user called Chris.

```sql
select s.username, n.name, t.value
from v$session s, v$statname n, v$sesstat t
where s.sid = t.sid
and t.statistic# = n.statistic#
and s.username = 'CHRIS';
```
Keep in mind that joining any view to V$SYSSTAT will give you system-wide performance information while joining any view to V$SESSTAT will give you performance information for an individual user.

**Sample DBA View Query**

Queries against the DBA views are also useful for performance tuning. For example, you may wish to know which tables have rows in them that are either chained or migrated. The query shown below would provide that information for a user named Chris whose schema has been recently analyzed using the ANALYZE SQL command.

```sql
select table_name, chain_cnt
from dba_tables
where owner = 'CHRIS'
and chain_cnt != 0;
```

Chaining and migration are discussed in more detail in Chapter 8, “Tuning Disk I/O.”

---

**Oracle-Supplied Tuning Utilities**

Oracle supplies several tuning utilities, in the form of scripts and PL/SQL packages, which can be used to help you identify possible database performance problems. Some commonly used Oracle-supplied tuning utilities are listed here:

- **UTLBSTAT.SQL/UTLESTAT.SQL** (Discussed in the following section)
- **STATSPACK** (Discussed in the following section)
- **UTLLOCKT.SQL** (Chapter 9, “Tuning Contention”)
- **CATBLOCK.SQL** (Chapter 9, “Tuning Contention”)
- **CATPARR.SQL** (Chapter 5, “Tuning the Database Buffer Cache”)
- **DBMSPOOL.SQL** (Chapter 4, “Tuning the Shared Pool”)
- **UTLCHAIN.SQL** (Chapter 8, “Tuning Disk I/O”)
- **UTLXPLAN.SQL** (Chapter 3, “SQL Application Tuning and Design”)

---
Sources of Tuning Information

Some of these scripts are run only once, to build tables or views that are useful for performance tuning. Other scripts are run every time you perform a tuning exercise. All of these scripts are discussed in detail in later chapters. However, because the output from the UTLBSTAT.SQL/UTLESTAT.SQL scripts and STATSPACK utility encompass all aspects of database performance, an introductory discussion of them is included here.

**UTLBSTAT.SQL and UTLESTAT.SQL**

Although largely replaced by the newer STATSPACK utility discussed in the next section, the Oracle-supplied **UTLBSTAT.SQL** and **UTLESTAT.SQL** scripts are still useful tools for Oracle Server performance tuning. The purpose of these scripts is to capture and summarize into a single report virtually all database and instance performance activity for a specific period of time. Since the process of running the **UTLBSTAT.SQL** and **UTLESTAT.SQL** scripts and the content of the resulting output is the same on all platforms and operating systems, it is still an important tool for you to understand.

The B in **UTLBSTAT.SQL** stands for *beginning*. The E in **UTLESTAT.SQL** stands for *ending*. As these names would imply, the **UTLBSTAT.SQL** script is run at the beginning of the tuning measurement period and **UTLESTAT.SQL** is run at the end of the tuning measurement period.

These scripts are located in `$ORACLE_HOME/rdbms/admin` on Unix systems and `%ORACLE_HOME%\rdbms\admin` on Windows 2000 systems.

**Running the UTLBSTAT.SQL Script**

Before running the **UTLBSTAT.SQL** script, it is important that the instance has been up long enough to allow the V$ dynamic performance views to be populated with meaningful statistics. Usually this means that you will want to wait and run the **UTLBSTAT.SQL** script after the database has been open for at least 30 minutes of normal user activity.

When the **UTLBSTAT.SQL** script is executed, it will do the following:

- First, it will drop any of the tables that the **UTLBSTAT.SQL** script is about to create—if they already exist. These tables may exist if a
previous execution of UTLBSTAT.SQL was not followed by the execution of UTLESTAT.SQL. Because the tables may not exist, it is normal to see ORA-00942 Table or View Does Not Exist errors during the execution of UTLBSTAT.SQL.

- Next, the UTLBSTAT.SQL script creates a series of temporary tables that are copies of the contents of some of the V$ views as they exist at that moment in time. For example, UTLBSTAT.SQL will create a table called stats$begin_event that is a copy of the v$system_event view.

The UTLBSTAT.SQL script is executed using the following command in SQL*Plus:

- Unix: @$ORACLE_HOME/rdbms/admin/utlbstat.sql
- Windows 2000: @%ORACLE_HOME%\rdbms\admin\utlbstat.sql

**Running the UTLESTAT.SQL Script**

A meaningful amount of time should be allowed to pass between the time that UTLBSTAT.SQL is run and the time UTLESTAT.SQL is run. Generally this duration should be at least 15 minutes, but could be several hours or even days.

If the instance is shut down between the time you run UTLBSTAT.SQL and UTLESTAT.SQL, your results will not be usable. This occurs because all the statistics in the V$ views that are used by these two scripts are cleared at instance shutdown and then repopulated at instance startup.

When the UTLESTAT.SQL script is executed, it will do the following:

- First, it will drop any of the tables that the UTLESTAT.SQL script is about to create—if they already exist. These tables may exist if a previous execution of UTLESTAT.SQL failed. Because the tables may not exist, it is normal to see ORA-00942 “Table or View Does Not Exist” errors during the execution of UTLESTAT.SQL.

- Next, the UTLESTAT.SQL script creates a series of temporary tables that are copies of the contents of the same V$ views that UTLBSTAT.SQL copied. This time, however, the tables have names like stats$end_event and will contain statistics as they exist at that moment in time.
Then, it calculates metrics between the statistics in the stats$begin tables, which were created by UTLBSTAT.SQL at the beginning of the tuning period, and the stats$end tables created by UTLESTAT.SQL at the end of the tuning period.

Finally, it generates a text file, REPORT.TXT, which contains the results of these calculations. This text file contains all the information needed for most database tuning activities.

The UTLESTAT.SQL script is executed using the following command in SQL*Plus:

Unix: @$ORACLE_HOME/rdbms/admin/utlestat.sql

Windows 2000: @%ORACLE_HOME%\rdbms\admin\utlestat.sql

The resulting REPORT.TXT file will be written to an operating system file in the user’s current directory.

The standard UTLBSTAT.SQL and UTLESTAT.SQL scripts both contain a CONNECT/AS SYSDBA string at the beginning of each script. If your database makes use of a password file, then you will have to edit each of these scripts and change the CONNECT/AS SYSDBA command to CONNECT SYS AS SYSDBA, or some other SYSDBA username. Then as the scripts are executed, they will prompt you for the SYSDBA password dynamically.

Interpreting the Contents of REPORT.TXT

Once generated by UTLBSTAT.SQL and UTLESTAT.SQL, the resulting REPORT.TXT file will contain all the tuning statistics for the period between the time UTLBSTAT.SQL was executed and the time when UTLESTAT.SQL was executed. REPORT.TXT is a simple text file that can be viewed with any editor. Upon initial examination, deciphering the contents of a REPORT.TXT file may seem like a daunting task. However, with practice and experience, you will quickly be able to spot problem areas in a lengthy REPORT.TXT file with a minimum of effort. References to relevant portions of the REPORT.TXT file will be made throughout the remainder of this book. A section from a sample REPORT.TXT file generated against an Oracle9i database on a Windows 2000 system is shown in Figure 2.2.
Statpack

Originally introduced in Oracle8i Release 2, STATSPACK is the updated and much improved cousin of the often-cryptic REPORT.TXT produced by the UTLBSTAT.SQL and UTLESTAT.SQL scripts discussed in the previous section. STATSPACK uses a combination of Oracle-supplied SQL scripts and PL/SQL packages to produce an evaluation of your database's overall performance and recommendations regarding possible performance-enhancing changes.
This is only a brief introduction to the STATSPACK utility. For full details see Chapter 21, “Using STATSPACK,” in the *Oracle9i Database Performance Guide and Reference Release 1 (9.0.1)* Part Number (A87503-02).

**Configuring STATSPACK**

Before using STATSPACK to monitor your database’s performance, the PERFSTAT schema that owns all the STATSPACK PL/SQL packages, tables, and indexes must be created. Executing the SPCREATE.SQL script while connected to the database as a SYSDBA user creates the PERFSTAT schema and all its associated objects.

The default password for the PERFSTAT schema is PERFSTAT. For security reasons, this password should be changed immediately after the PERFSTAT user is created.

The SPCREATE.SQL script is executed using the following command in SQL*Plus:

- Unix: `@$ORACLE_HOME/rdbms/admin/spcreate.sql`
- Windows 2000: `@%ORACLE_HOME%\rdbms\admin\spcreate.sql`

During the execution of the SPCREATE.SQL script, you will be prompted for the names of the default and temporary tablespaces that should be assigned to the PERFSTAT user. In order to manage the STATSPACK segments more easily, a new 250MB tablespace just for the purpose of holding the PERFSTAT segments should be created. The PERFSTAT segments initially require only about 80MB of tablespace, but will require more over time, as additional performance statistics are stored. The SPCREATE.SQL script generates three log files, SPCUSR.LIS, SPCTAB.LIS and SPCPKG.LIS, which should be examined after execution to confirm that all the STATSPACK components were properly created.

Do not use the SYSTEM tablespace as the default or temporary tablespace for the PERFSTAT schema. Doing so will cause the SPCREATE.SQL script to fail. If the script does fail during execution, you can run the SPDROP.SQL script while logged in as a SYSDBA to drop the partially created PERFSTAT schema.
Collecting Performance Statistics Using STATSPACK

Once the STATSPACK components are configured, you can use them to monitor your database’s performance. STATSPACK performance monitoring can be done in two ways:

- Manually executing the STATSPACK.SNAP procedure
- Automatically using the SPAUTO.SQL scheduling script

Collect Statistics Using Manual STATSPACK Execution

To manually initiate the collection of database performance statistics using the STATSPACK utility, you must execute the SNAP procedure in the STATSPACK PL/SQL package by issuing the following command while logged into SQL*Plus as the PERFSTAT user:

```
SQL> execute statspack.snap
```

When the SNAP procedure is executed, a snapshot of the current values for many database performance statistics are captured and stored in the tables that reside in the PERFSTAT schema. These statistics will then be compared to the performance statistics gathered at a later point in time when the STATSPACK.SNAP procedure is executed a second time. However, unlike UTLBSTAT.SQL and UTLESTAT.SQL discussed in the previous section, the STATSPACK statistics are not deleted after the ending statistics are gathered. Instead, the statistics from a third, fourth, or infinite number of snapshots can be captured and stored indefinitely in the PERFSTAT schema tables. Performance statistics can then be generated between any two points in time when snapshots were taken.

Collect Statistics Using Automatic STATSPACK Execution

Since the STATSPACK utility has the ability to collect many performance statistic snapshots, you may want to schedule snapshots at regular intervals throughout the day, week, or month. Oracle provides an SQL script just for this purpose. The script, called SPAUTO.SQL, uses Oracle’s DBMS_JOBS package to schedule the snapshot statistic data collection.
Chapter 2 • Sources of Tuning Information

The `init.ora` parameter `JOB_QUEUE_PROCESSES` must be set to a non-zero value in order for automatic snapshot statistic collection to function. This parameter tells Oracle how many background processes to use when managing jobs scheduled with the `DBMS_JOBS` package.

The `SPAUTO.SQL` script is executed using the following command in SQL*Plus:

- Unix: `@$ORACLE_HOME/rdbms/admin/spauto.sql`
- Windows 2000: `@%ORACLE_HOME%\rdbms\admin\spauto.sql`

By default, the `SPAUTO.SQL` script will set up snapshot collection jobs to run every hour, on the hour. You can change the frequency of the snapshot statistic collection period by editing the `SPAUTO.SQL` script and changing the submission times shown in bold in Figure 2.3. After making these changes, re-executing the `SPAUTO.SQL` script will cause statistic collection to occur on the schedule you specified.

**FIGURE 2.3** Changing the STATSPACK statistics collection interval

```
Rem

spool spauto.log
-- Schedule a snapshot to be run on this instance every hour, on the hour
variable jobno number;
variable instno number;
begin
  select instance_number into :instno from v$instance;
  dbms_job.submit( :jobno, 'statspack.snap', trunc(SYSDATE+1/24, 'HH'), 'trunc(SYSDATE+1/24, 'HH')', true, :instno);
  commit;
end;
/
```

**Generating STATSPACK Tuning Reports**

Once the STATSPACK utility has been used to collect two or more snapshots of database performance statistics, a report showing tuning recommendations can be generated using the `SPREPORT.SQL` script. The `SPREPORT.SQL` script is executed using the following command in SQL*Plus:

- Unix: `@$ORACLE_HOME/rdbms/admin/spreport.sql`
- Windows 2000: `@%ORACLE_HOME%\rdbms\admin\spreport.sql`
During the script’s execution you will be prompted for three parameters. Each of these parameters is shown in bold in Figure 2.4:

- The ID of the snapshot that you would like to use as a starting point for your performance analysis (2 in Figure 2.4).
- The ID of the snapshot that you would like to use as an ending point for your analysis (4 in Figure 2.4).
- The name you would like to give to the text file that will contain the results of the performance report (statspack.txt in Figure 2.4).

**FIGURE 2.4** STATSPACK report generation parameters

<table>
<thead>
<tr>
<th>Instances in this Statspack schema</th>
</tr>
</thead>
<tbody>
<tr>
<td>DB Id</td>
</tr>
<tr>
<td>-------</td>
</tr>
<tr>
<td>S08515328</td>
</tr>
</tbody>
</table>

Using 500515328 for database Id
Using 1 for instance number

Completed Snapshots

<table>
<thead>
<tr>
<th>Instance</th>
<th>DB Name</th>
<th>Snap Id</th>
<th>Snap Started</th>
<th>Level</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>prod</td>
<td>PROD</td>
<td>1</td>
<td>09 Jan 2002 19:41</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>09 Jan 2002 19:52</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>09 Jan 2002 20:32</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td>09 Jan 2002 20:33</td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>

Specify the Begin and End Snapshot Ids

Enter value for begin_snap: 2
Begin Snapshot Id specified: 2

Enter value for end_snap: 4
End Snapshot Id specified: 4

Specify the Report Name

The default report file name is sp_2.4. To use this name, press <return> to continue, otherwise enter an alternative.

Enter value for report_name: statspack.txt
Interpreting the Contents of the STATSPACK Report

Once generated by the SPREPORT.SQL script, the resulting report will contain all the tuning statistics for the period between the two points in time when the performance snapshots were taken. The resulting report file is a simple text file that can be viewed with any editor. Unlike the contents of the REPORT.TXT file generated by the UTLBSTAT.SQL and UTLESTAT.SQL scripts, the STATSPACK report is very clearly laid out, making it easy to identify the important tuning areas. References to relevant portions of the STATSPACK report file will be made throughout the remainder of this book. A section from a sample STATSPACK report file generated against an Oracle9i database on Windows 2000 system is shown in Figure 2.5.

**FIGURE 2.5** Sample section of STATSPACK report file

<table>
<thead>
<tr>
<th>STATSPACK report for</th>
</tr>
</thead>
<tbody>
<tr>
<td>DB Name</td>
</tr>
<tr>
<td>---------</td>
</tr>
<tr>
<td>PROD</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>snap id</th>
<th>snap Time</th>
<th>sessions</th>
<th>curs/sess</th>
<th>comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Begin Snap:</td>
<td>2 09-Jan-02 19:11:08</td>
<td>9</td>
<td>5.3</td>
<td></td>
</tr>
<tr>
<td>End Snap:</td>
<td>4 09-Jan-02 19:12:05</td>
<td>9</td>
<td>3.3</td>
<td></td>
</tr>
<tr>
<td>Elapsed:</td>
<td>10.97 (mins)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Cache Sizes (end)

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~ Buffer Cache: 47M  Std Block Size: 1KB
~~ shared pool size: 64M  Log buffer: 1,024K
```

Load Profile

```
Redo size: 553.74  122.584.57
Logical reads: 4.83  1.066.33
Block changes: 0.93  204.33
Physical reads: 0.22  49.00
Physical writes: 0.23  51.00
User calls: 0.83  181.33
Parses: 0.12  27.00
Hard parses: 0.08  5.67
Sorts: 0.15  32.67
Logons: 0.00  0.07
Execs: 0.12  47.33
Transactions: 0.00

% Blocks changed per read: 19.27  recursive call %: 62.77
Rollback per transaction %: 0.00  rows per sort: 63.49
```

Instance Efficiency Percentages (Target 100%)

```
Buffer waits %: 100.00  redo waits %: 100.00
Buffer hit %: 100.00  in-memory sort %: 97.95
Library hit %: 89.11  soft parse %: 79.01
Execute to parse %: 42.96  latch hit %: 100.00
Parse CPU to Parse elapsed %: % non-parse CPU:
```

Shared Pool Statistics

```
<table>
<thead>
<tr>
<th>Memory Usage</th>
<th>% SQL with executions</th>
<th>% Memory for SQL w/execs</th>
</tr>
</thead>
<tbody>
<tr>
<td>27.49</td>
<td>39.04</td>
<td>63.47</td>
</tr>
<tr>
<td>28.35</td>
<td>39.42</td>
<td>79.74</td>
</tr>
</tbody>
</table>
```

<table>
<thead>
<tr>
<th>Memory</th>
<th>disk</th>
<th>non-disk</th>
<th>total</th>
</tr>
</thead>
<tbody>
<tr>
<td>1024</td>
<td>1024</td>
<td>2048</td>
<td>2048</td>
</tr>
</tbody>
</table>
Graphical Performance Tuning Tools

Oracle provides several Graphical User Interface (GUI) database management tools that can be used to aid a tuning effort. These GUI tools are divided into three product categories:

- Oracle Enterprise Manager Console
- Oracle Diagnostics Pack
- Oracle Tuning Pack

The Oracle Enterprise Manager Console is included with the basic Oracle9i database software. The Diagnostics and Tuning Packs are extra-cost options that require a separate license with Oracle. Each of these GUI tools is discussed briefly in the following section. Additional details about some of these GUI tools also appear in relevant sections of subsequent chapters.

Free evaluation copies of Oracle’s Diagnostic and Tuning Packs can be downloaded from http://technet.oracle.com/software/products/oem/content.html.

Oracle Objective

Collect statistics using Enterprise Manager

Oracle Enterprise Manager Console

Oracle Enterprise Manager (OEM) is the core of Oracle’s GUI database management tools. The OEM Console includes several tools designed to help you graphically monitor and manage your databases. Figure 2.6 shows a picture of the OEM Console for Oracle9i.

The OEM Console seamlessly integrates components of the DBA Management, Tuning, and Diagnostic Packs into one centralized GUI environment from which you can administer all your databases. The base OEM product is made up of six management tools. These tools and their uses are summarized in Table 2.6.

A critical component in the OEM architecture is the Oracle Intelligent Agent. This autonomous OS process allows the OEM Console to perform a variety of tasks against the databases that the Console is managing.
## FIGURE 2.6 Oracle9i Enterprise Manager Console

![Oracle9i Enterprise Manager Console](image)

## TABLE 2.6 Components of the Base Oracle Enterprise Manager Console

<table>
<thead>
<tr>
<th>Component</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instance Manager</td>
<td>Starting, stopping, and managing Oracle instances</td>
</tr>
<tr>
<td>Schema Manager</td>
<td>Creating and managing Oracle database objects.</td>
</tr>
<tr>
<td>Security Manager</td>
<td>Creating and managing Oracle users, privileges, and roles</td>
</tr>
<tr>
<td>Storage Manager</td>
<td>Creating and managing tablespaces and datafiles</td>
</tr>
<tr>
<td>Replication Manager</td>
<td>GUI environment for managing Oracle snapshots and replication features</td>
</tr>
<tr>
<td>Workspace Manager</td>
<td>Virtual workspace that allows changes to be made to the data in the database.</td>
</tr>
</tbody>
</table>
The Oracle Intelligent Agent executes independently of the OEM Console. One Intelligent Agent process runs on each server (or node) that the OEM tool is monitoring. There is only one agent on each server node that OEM is monitoring regardless of how many databases exist on that node. The Intelligent Agent and the databases it monitors represent the third tier in Oracle’s three-tier OEM architecture. The middle tier of the OEM architecture is the Oracle Management Server (OMS), which acts as the repository for the OEM configuration. The OMS is accessible via the OEM Console from a variety of clients including Unix or Windows 2000 workstations and web browsers. These clients represent the first tier of the OEM architecture. An overview of the OEM three-tier architecture model is shown in Figure 2.7.

In addition to the Intelligent Agent, Oracle8i also required that a second process, called Data Gatherer, be running in order to use some GUI performance tuning tools. In Oracle9i, the activities of Data Gatherer have been incorporated into the Intelligent Agent instead.
The role of the Intelligent Agent is to carry out the actions specified within the OEM Console. These actions can take on two forms: scheduled jobs and event monitoring.

**Scheduled Jobs**  The Intelligent Agent and the OEM Console can be used to schedule and perform jobs against the databases on the server node that the Intelligent Agent manages. The time, date, and name of the job to be executed are scheduled using the Jobs pane of the OEM Console. Once scheduled, the Intelligent Agent is the process that actually executes the job at the appointed time. Nightly backups, purging of old archive logs, and database statistics gathering are all example of jobs that can be scheduled using the Intelligent Agent.

**Event Monitoring**  OEM includes a sophisticated event management system. These events can be very useful tools for performance tuning. For example, you can use OEM events and the Intelligent Agent to notify you via e-mail or paging whenever a table containing chained rows is detected in the database (row chaining is discussed in Chapter 8, “Tuning Disk I/O”).

Event monitoring can be configured so that whenever the Intelligent Agent detects that a particular event has occurred, a specified task, known as a *Fixit Job*, is performed in response to that event. Therefore, problems that would normally require manual intervention can be fixed immediately according to your predefined specifications.

Oracle supplies a number of predefined events in the OEM Console so that you can immediately start monitoring for common events that are of interest to you. These events include monitoring whether the database is up or down, whether the Oracle Net Listener is up or down, and whether the server node that the database resides on is up or down. When extra-cost options like the Tuning or Diagnostic Packs are added, they provide additional built-in monitoring for events like excessive extent growth and low tablespace storage availability, as well as memory utilization statistics and contention for certain database resources. You can also create your own custom events for the Intelligent Agent to monitor by using the TCL command language.
Additional information on using TCL scripting to create custom OEM events can be found in Chapter 4, “Job and Event Scripts,” in the Oracle Intelligent Agent Users Guide, Release 9.0.1 (Part Number A88771-02).

Real World Scenario
Securing the Intelligent Agent Oracle Account

You have just configured the components of the Enterprise Manager infrastructure. You’re concerned that the user account (DBSNMP) used by the Intelligent Agent to access the managed databases is a security risk because of its well documented default password of DBSNMP. You document the steps to change the default password and produce the following checklist:

1. Delete all OEM Jobs or Events associated with the database whose DBSNMP password you are changing.

2. Shut down the Intelligent Agent on the node hosting the database whose DBSNMP password you are changing.

3. Edit the SNMP_RW.ORA file located in $ORACLE_HOME/network/admin or Unix systems and %ORACLE_HOME%\network\admin on Windows 2000 systems. Add this line to the SNMP_RW.ORA file - where ConnectString is the Oracle Net connect string for the database whose DBSNMP password you are changing:

   SNMP.CONNECT.ConnectString.PASSWORD=Password

4. Edit the CATSNMP.SQL file found in $ORACLE_HOME/rdbms/admin on Unix systems and %ORACLE_HOME%\rdbms\admin on Windows 2000 systems. Change the reference to the default password at the end of this file - identified by DBSNMP; - to the new password you assigned in the previous step. Changing this file prevents the DBSNMP password from being inadvertently reset back to the default value of DBSNMP when the CATSNMP.SQL script is re-run at a later time.
Oracle Diagnostics Pack

The Oracle Diagnostics Pack is made up of several tools designed to help you identify, diagnose, and repair problem areas in your databases. Table 2.7 shows the components of the Diagnostics Pack.

<table>
<thead>
<tr>
<th>TABLE 2.7 Components of the Diagnostics Pack</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Component</strong></td>
</tr>
<tr>
<td>Capacity Planner</td>
</tr>
<tr>
<td>Performance Manager</td>
</tr>
<tr>
<td>TopSessions</td>
</tr>
<tr>
<td>Trace Data Viewer</td>
</tr>
</tbody>
</table>

5. Set the security on the SNMP_RW.ORA and CATSNMP.SQL files at the OS level so that the password is protected. Do this on Unix systems by setting the file permission mask on the files to 700, and by making the files accessible only to Administrators on Windows 2000 systems.

6. Log in as a SYSDBA user to the database whose DBSNMP password you are changing. Change the DBSNMP user’s password using: ALTER USER DBSNMP IDENTIFIED BY *Password*;

7. Restart the Intelligent Agent. If all the changes were made correctly the Intelligent Agent should start without any errors. If errors occur, you can examine the Intelligent Agent log and trace files for errors. These logs include the nmi.conf.log and nmi.log files located in $ORACLE_HOME/network/log on Unix systems and %ORACLE_HOME%\network\log on Windows 2000 systems, and the trace file located in $ORACLE_HOME/network/trace/dbsnmp.trc on Unix systems and %ORACLE_HOME%\network\trace\dbsnmp.trc on Windows 2000 systems.
With the exception of Capacity Planner, each of these GUI tools is very useful for performance tuning. Each of these tools is described in more detail in the following sections.

**Performance Manager**

*Performance Manager* is a GUI performance monitoring tool that allows you to observe database performance indicators in your databases in real time. The indicators Performance Manager tracks are the same ones that we will discuss throughout this book. However, rather than issuing queries or inspecting STATSPACK or REPORT.TXT files for this information, it is presented visually in the form of charts and graphs. Figure 2.8 shows a sample of Performance Manager’s graphical output.

One unique feature of Performance Manager is that the performance data can be recorded for several hours and then graphically “played back” at an accelerated rate in order to help you identify spikes in activity that may be negatively impacting your system. Table 2.8 shows some of the tuning areas you can monitor using the predefined Performance Manager charts. Notice that many of these areas are the same as those covered from a non-Performance Manager perspective in subsequent chapters.

In addition to these specific areas, Performance Manager also offers several predefined charts that summarize the overall performance of the server into one easy-to-interpret chart. User-defined charts can also be created.
Sources of Tuning Information

**FIGURE 2.8** Performance Manager Output

**TABLE 2.8** Performance Manager Monitoring Areas

<table>
<thead>
<tr>
<th>Area</th>
<th>What Can Be Monitored</th>
</tr>
</thead>
<tbody>
<tr>
<td>Disk I/O</td>
<td>File I/O, File I/O rate, and System I/O</td>
</tr>
<tr>
<td>Contention</td>
<td>Latches, Shared Server Queues, Redo activities</td>
</tr>
<tr>
<td>Instance</td>
<td>Statistics for the overall system, individual processes,</td>
</tr>
<tr>
<td></td>
<td>and individual users</td>
</tr>
<tr>
<td>Load</td>
<td>Sort and throughput activity</td>
</tr>
<tr>
<td>Memory</td>
<td>Hit ratios for all SGA components</td>
</tr>
<tr>
<td>Resources</td>
<td>Top resource users</td>
</tr>
</tbody>
</table>
**TopSessions**

*TopSessions* is a graphical tool for identifying in real time which users are consuming the most server resources. Resources that can be monitored include CPU, memory, file I/O, and transaction volume. Figure 2.9 shows a sample TopSessions screen.

![TopSessions Screen](image)

TopSessions is also useful for identifying locking issues, which may cause excessive waits for other database users. Locking is discussed in detail in Chapter 9, “Tuning Contention.”

**Trace Data Viewer**

Many Oracle servers are accessed by applications that are not Oracle-centric in nature. These applications only use the Oracle database as a repository for the application’s data. All the application logic and processing happens outside the database. Gathering and analyzing performance data from these types of applications can be difficult. Oracle’s *Trace Data Viewer* GUI tool allows you to collect performance data for specific events within these types of applications. This is accomplished through the use of an Application
Program Interface (API). An API designed specifically for this purpose has been incorporated in the Oracle Server since Oracle 7.3. The types of data that can be collected via this API generally relate to resource usage, execution times, and application specific data. Additionally, you can also use Trace Data Viewer to gather performance information related to regular database events.

See Chapter 12, “Using Oracle Trace,” in the Oracle9i Database Performance Guide and Reference, Release 1 (9.0.1) (Part Number A87503-02) for more details on configuring, scheduling, and administering the Oracle Trace Manager GUI utility.

Lock Monitor

Lock Monitor is a very useful tool for identifying which user sessions are locking database segments during processing. Figure 2.10 shows a sample Lock Monitor screen.

FIGURE 2.10 The Lock Monitor
The output in Figure 2.10 shows that the user JJJOHNSON is trying to obtain an exclusive lock on the EMPLOYEES table, but is currently waiting to obtain that lock because the user SYSTEM is blocking his request.

A complete discussion of performance tuning issues related to locking can be found in Chapter 9, “Tuning Contention.”

**Top SQL**

Top SQL is designed to help you easily identify which SQL statements are consuming the most system resources. Figure 2.11 shows a sample of the Top SQL screen.

![Top SQL screen](image)

The output in Figure 2.11 shows the SQL commands that have consumed the most system resources. Top SQL also allows you to drill down on any of the statements simply by double-clicking on that statement. The drilldown information would include details like the statement’s execution plan and detailed statistics about the statements I/O and CPU usage.
A complete discussion of SQL statement tuning can be found in Chapter 3.

## Performance Overview

Performance Overview is a quick one-stop location for a broad overview of the current performance of the database. Figure 2.12 shows a sample of the Performance Overview screen.

**Figure 2.12** Performance Overview Screen

The output in Figure 2.12 shows the performance statistics for memory and CPU usage as well as physical I/O and SQL processing.
A complete discussion of each of the areas represented on the Performance Overview summary screen is included in the remaining chapters of this book.

Oracle Tuning Pack

The Oracle Tuning Pack is designed specifically to assist you with database performance tuning issues. Table 2.9 shows the components of the Tuning Pack.

**TABLE 2.9** Components of the Tuning Pack

<table>
<thead>
<tr>
<th>Component</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Oracle Expert</td>
<td>Wizard-based database tuning tool based on the Oracle Tuning Methodology.</td>
</tr>
<tr>
<td>SQL Analyze</td>
<td>Analyzes and rewrites SQL statements to improve their performance.</td>
</tr>
<tr>
<td>Index Tuning Wizard</td>
<td>Helps identify which indexes are being utilized by application users, and the effectiveness of those indexes.</td>
</tr>
<tr>
<td>Outline Management</td>
<td>Used to manage Stored Outlines. Stored Outlines are discussed in Chapter 3, “SQL Application Tuning and Design.”</td>
</tr>
<tr>
<td>Reorg Wizard</td>
<td>Assists with table and index reorganization tasks like moving segments to new tablespaces and rebuilding segments with new sizing specifications.</td>
</tr>
<tr>
<td>Tablespace Map</td>
<td>Graphically shows you were each segment's individual extents are stored within a Tablespace's datafiles.</td>
</tr>
</tbody>
</table>

While all of these GUI tools provide important tuning information, Oracle Expert, SQL Analyze, and Index Tuning Wizard are particularly useful for database-level performance tuning.
Oracle Expert

Oracle Expert is a GUI tool that gathers and analyzes performance tuning data according to your specifications. Once the data is analyzed, Oracle Expert then uses its built-in inference engine to formulate tuning recommendations. These recommendations can be presented in a summary report along with SQL scripts and new init.ora files that can be used to help you implement the recommended changes.

The Oracle Expert tuning methodology consists of the following steps:

1. Define the tuning scope
2. Collect data
3. View and edit the collected data and adjust tuning rules
4. Analyze the collected data
5. Generate tuning recommendations and review the results
6. Create SQL scripts needed to implement the recommendations

Each of these steps is represented by a different tab on the Oracle Expert interface. For example, Figure 2.13 shows the Scope tab where the overall objectives of the tuning effort are specified.

FIGURE 2.13 Oracle Expert
The recommendation summary generated by the Oracle Expert tool is verbose, offering justifications for each tuning change that is recommended. Figure 2.14 shows a sample output from an Oracle Expert recommendation report.

**FIGURE 2.14** Oracle Expert recommendation report

Like the Top SQL GUI tool found in the Diagnostics Pack, SQL Analyze is also designed to help you identify poorly performing SQL statements. Figure 2.15 shows a sample SQL Analyze screen.
FIGURE 2.15 SQL Analyze screen

The SQL Analyze tool offers several useful features including the ability to store and view historical SQL statements in a repository, generate Explain Plans, and produce HTML reports of its findings.

A complete discussion of the generation and interpretation of Explain Plans can be found in Chapter 3.

Index Tuning Wizard

During application development, programmers frequently identify table columns that make good candidates for indexing. However, not all indexes that were placed on tables during the design stage are actually used when the application is put into use. While indexes are designed to improve the performance of queries, having unused indexes in an application can actually hinder performance. The Index Tuning Wizard GUI tool is designed to help you identify which application indexes are not being used so that you can consider dropping them. Figure 2.16 shows a sample Index Tuning Wizard screen.
The Index Tuning Wizard will not only help identify unused indexes, but will also write the SQL statements needed to drop or modify them in order to enhance performance.

See the author’s article, “Using New Oracle9i Features for Managing Indexes,” in the January/February 2002 issue of Oracle Magazine (http://www.oracle.com/oramag/oracle/index.html) for details on how to determine which application indexes are underutilized.

Summary

Having accurate and complete information is critical before, during, and after the tuning process. This information can be gathered from several locations. The Alert log contains a variety of instance-wide event information, while the background and user trace files can be used to more narrowly
identify what is happening in the database. Oracle’s dynamic V$ and DBA data dictionary views also allow you to gather important statistics and information that is needed for database tuning.

Oracle also offers two utilities, STATSPACK and UTLBSTAT.SQL/UTLESTAT.SQL, which you can use to capture statistics for a determinate period of time and then analyze those statistics using the resulting STATSPACK report and REPORT.TXT output.

The Oracle Enterprise Manager Console also provides useful tuning information through its event monitoring via the Intelligent Agent. Several useful performance tuning add-on packs are also available for the Oracle Enterprise Manager. These include the Diagnostic and Tuning Packs, both of which provide additional performance tuning capabilities through the use of the Performance Manager and Oracle Expert components.

Exam Essentials

Know how to use dictionary and performance views. Understand the names and contents of the V$ dynamic performance views and the DBA data dictionary views that contain tuning information.

Describe how to install and use STATSPACK. Be able to describe how to install and use the STATSPACK utilities to generate tuning recommendations.

Know what other Performance Tuning tools are available. List and describe the tuning-related components of the Oracle Enterprise Manager GUI tool and script-based tuning tools like UTLBSTAT.SQL and UTLESTAT.SQL.

Understand locations and contents of Alert and Trace Files. Know where to find the Alert and Trace files generated by the instance. Be able to describe the tuning information found in Alert logs, background Trace files, and user Trace files.
Before you take the exam, be certain you are familiar with the following terms:

Alert log

BACKGROUND_DUMP_DEST

data dictionary views

database event

dynamic performance views

Fixit Job

Graphical User Interface (GUI)

MAX_DUMP_FILE_SIZE

Optimal Flexible Architecture (OFA)

Oracle Diagnostics Pack

Oracle Enterprise Manager (OEM)

Oracle Expert

Oracle Intelligent Agent

Performance Manager

SQL Analyze

STATSPACK

TopSessions

Trace Data Viewer

trace files

USER_DUMP_DEST

UTLBSTAT.SQL

UTLESTAT.SQL
Review Questions

1. The Alert log contains information on all but which of the following:
   A. Instance startup and shutdown.
   B. Creation of new database users.
   C. Redo log switches.
   D. All of the above are in the Alert log.

2. Which of the following views can be used to determine which user generated a particular user trace file? (Choose all that apply.)
   A. V$SYSTEM
   B. V$SESSION
   C. V$USER
   D. V$PROCESS

3. Which of the following init.ora parameters activates tracing at the instance level?
   A. TRACE_ON
   B. TRACE_SQL
   C. SQL_FILE
   D. SQL_TRACE

4. Which of the following Oracle-supplied PL/SQL packages can be used to activate tracing in another user's session?
   A. DBMS_SYSTEM
   B. DBMS_SESSION
   C. DBMS_TRACE
   D. DBMS_UTILITY

5. Which of the following views contains information about overall system statistics, both previously and currently connected?
A. V$SGASTAT
B. V$STATNAME
C. V$SESSTAT
D. V$SYSSTAT

6. Which DBA view would show storage, row, and block information for an individual table?
   A. DBA_TABLES
   B. DBA_OBJECTS
   C. DBA_TABLE_SIZE
   D. DBA_TAB_PRIVS

7. What is the name of the file that is generated by UTLBSTAT.SQL and UTLESTAT.SQL?
   A. REPORT.TXT
   B. REPORT.LOG
   C. REPORT.LST
   D. REPORT.SQL

8. The autonomous process that runs on each Oracle server to manage jobs and monitor events on that node is called:
   A. Oracle Net Listener
   B. Enterprise Manager Console
   C. Intelligent Agent
   D. Data Gatherer

9. The OEM GUI tool that records and reports database performance statistics graphically in real time is called:
   A. TopSessions
   B. Performance Manager
   C. Trace Data Viewer
   D. Capacity Planner
10. What is the GUI tool that gathers performance data, analyzes that data, and then makes appropriate tuning recommendations called?
   A. SQL Analyze
   B. Oracle Expert
   C. Trace Expert
   D. SQL Expert

11. When you are using UTLBSTAT/UTLESTAT, what should you avoid doing if you want accurate statistics?
   A. Disconnect from the process that started the UTLBSTAT utility.
   B. Stop and Start the Oracle Instance.
   C. Stop the Oracle Agent.
   D. None of these cause problems with this utility.

12. What is the name of the Schema that owns all of the objects that are used by the STATSPACK utility?
    A. PERFSTAT
    B. DBSNMP
    C. SYS
    D. SYSTEM

13. What is an advantage of STATSPACK versus UTLBSTAT/UTLESTAT?
    A. Statistics are not deleted after each run of the process.
    B. Performance statistics can then be generated between any two points in time when snapshots were taken.
    C. The STATSPACK report is much clearer than the output from UTLBSTAT/UTLESTAT.
    D. These are all advantages.
14. What does the SPAUTO.SQL script perform?
   A. Generates the STATSPACK report
   B. Manual method of gathering database statistics with STATSPACK
   C. Automated method of gathering database statistics with STATSPACK
   D. None of the above

15. Which of the following Oracle Enterprise Manager components can generate a listing of tuning recommendations?
   A. The Intelligent Agent
   B. Performance Manager
   C. Oracle Expert
   D. Oracle Management Server

16. What is the maximum number of intelligent agent processes that can be running on each server node that Enterprise Manager is monitoring?
   A. One
   B. Two
   C. Four
   D. Dependent on the number of CPU’s

17. “Monitors user session activities including resource consumption and locking” best describes what utility?
   A. Capacity Planner
   B. TopSessions
   C. Trace Data Viewer
   D. Performance Manager
18. A developer has written a query that is not optimally tuned. The DBA could use what utility to assist the developer in tuning this query?
   A. STATSPACK
   B. Capacity Planner
   C. UTLBSTAT/UTLESTAT
   D. SQL Analyze

19. A DBA has performed system monitoring and has some ideas on changes to the database that could be made to enhance system performance. What utility could be used to support her findings and add credence to her recommendations?
   A. STATSPACK
   B. Oracle Expert
   C. REPORT.TXT
   D. Intelligent Agent

20. What user does the Intelligent Agent connect to the database with?
   A. DBSNMP
   B. SYS
   C. SYSTEM
   D. PERFSTAT

21. What GUI tool can be used to monitor which users are issuing the most resource-intensive SQL statements?
   A. Top SQL
   B. Lock Monitor
   C. Index Tuning Wizard
   D. Trace Data Viewer
Answers to Review Questions

1. B. The Alert log does not contain any user information.

2. B, D. Joining the V$SESSION and V$PROCESS views allows you to associate a username with a process ID. This process ID is included in the user's trace file name.

3. D. Setting the init.ora parameter SQL_TRACE = TRUE will activate tracing for every session that connects to the instance.

4. A. The SET_SQL_TRACE_IN_SESSION procedure in DBMS_SYSTEM is used to activate tracing in another user's session using the user's sid and serial# from V$SESSION.

5. D. V$SYSSTAT shows overall system statistics for all sessions both current and historical, since instance startup.

6. A. Once analyzed, a table's storage utilization, blocks usage, and row length information can be found in DBA_TABLES.

7. A. The UTLESTAT.SQL script generates the REPORT.TXT file by performing calculations on the stat$begin tables created by UTLBSTAT.SQL and the stats$end tables created by UTLESTAT.SQL.

8. C. The Oracle Intelligent Agent runs on each node that contains an Oracle database. The agent monitors the databases on that node on behalf of the Oracle Enterprise Manager Console.

9. B. The OEM Performance Manager GUI tool displays database performance data in a graphical format.

10. B. The Oracle Expert tool analyzes database statistics according to your specifications before making tuning recommendations.

11. B. When using the UTLBSTAT/UTLESTAT utilities, you should not stop and restart the instance. This will invalidate any of the statistics that have been gathered.

12. A. The PERFSTAT schema is the owner of all of the objects that are used by the STATSPACK utility.
13. D. These are all advantages of STATSPACK over UTLBSTAT/UTLESTAT. Because this information is kept in an Oracle database and not deleted between executions of the STATSPACK process, it can be queried and kept for any duration of time. Also, the reports are much easier to understand.

14. C. SPAUTO.SQL provides an automated way to execute the STATSPACK utility. The script uses Oracle’s DBMS_JOBS package to schedule the snapshot statistic data collection.

15. C. The Oracle Expert OEM component can generate a lengthy report containing tuning recommendations for the areas that were selected for analysis.

16. A. The maximum number of intelligent agent process per Oracle Server node is one.

17. B. The TopSessions utility comes as part of the Oracle Enterprise Manager Diagnostics Pack and can be used to monitor user session activities including resource consumption and locking.

18. D. The DBA would use the SQL Analyze utility to help the developer with the query. This utility is especially useful for tuning of problematic SQL statements.

19. B. The Oracle Expert utility can be used as a tool to help a DBA in uncovering performance bottlenecks; it also makes recommendations of changes to the system to enhance performance. This can be a valuable tool to use to validate a DBA’s own analysis.

20. A. The Intelligent Agent process on each node connects to the databases on that node as the user DBSNMP.

21. A. The Top SQL component of the Oracle Enterprise Manager Diagnostics pack shows which application users are consuming the most server resources.
SQL Application Tuning and Design

ORACLE9i PERFORMANCE TUNING EXAM OBJECTIVES COVERED IN THIS CHAPTER:

✓ Describe the role of the DBA in tuning applications
✓ Explain different storage structures and why one storage structure might be preferred over another
✓ Explain the different types of indexes
✓ Explain Index Organized Tables
✓ Describe partitioning methods
✓ Explain the use of the DBMS_STATS procedure
✓ Describe Materialized Views and the use of Query Rewrites
✓ List requirements for OLTP, DSS, and Hybrid Systems
✓ Describe how the Optimizer is used
✓ Explain the concept of plan stability
✓ Use stored outlines
✓ Describe how hints are used
✓ Collect statistics on indexes and tables
✓ Describe the use of histograms
✓ Copy statistics between databases
✓ Perform index reorganization
✓ Monitor indexes to determine usage

NOTE
Exam objectives are subject to change at any time without prior notice and at Oracle’s sole discretion. Please visit Oracle’s Certification website (http://www.oracle.com/education/certification/) for the current exam objectives listing.
Problems with poorly performing applications can frequently be traced to badly structured SQL statements or inefficient database design. Because of this, Oracle provides you with several tools to help you identify and remedy poorly performing application SQL.

These tools can be very valuable on development systems where application code is still being refined. However, even on production systems, where Oracle’s newer tuning principles recognize that tuning application SQL or design is difficult or impossible, these tools may help you identify SQL statements that might benefit from more broad DBA intervention—like adding an index to improve the performance of a long-running report query.

**Oracle Objective**

Describe the role of the DBA in tuning applications

Tuning modifications of this type require that you understand all the tuning options available to you, from SQL hints and stored outlines to materialized views and index-organized tables. You must also consider whether the tools available to you are appropriate given the type of system that is being evaluated. For example, Online Transaction Processing Systems have different SQL and design considerations than do Decision Support Systems.

However, no matter the type of system, or what type of tuning problem you face, you must first accurately measure the performance of the current application SQL before identifying areas of possible improvement.

**Measuring SQL Performance**

Effective database tuning requires that you take accurate and timely measurements of specific SQL activity occurring within the application.
Once these measurements are taken, you can attempt to improve their performance using a variety of techniques. A complete discussion of SQL statement tuning techniques is beyond the scope of this book and the Performance Tuning OCP exam. However, as a DBA, it is important for you to understand the five most commonly used tools for measuring the performance of SQL statements: the TKPROF utility, Top SQL, Explain Plans, AUTOTRACE, and STATSPACK reports.

The **TKPROF Utility**

The Trace Kernel Profile, or **TKPROF**, utility is used to format user trace files that were generated using one of the user session tracing techniques discussed in Chapter 2, “Sources of Tuning Information.” User trace files contain all the SQL statements that the user issued while tracing was active, information about how long each statement took to execute, and the number of resources the statement consumed.

<table>
<thead>
<tr>
<th>Oracle Objective</th>
<th>Use SQL Trace and TKPROF</th>
</tr>
</thead>
</table>

Because the contents of a raw, unformatted trace file are difficult to interpret, the TKPROF utility is used to convert the trace file into a form you can more easily understand. The following sections explain how to use this utility.

**Formatting a User Trace File Using **TKPROF**

The TKPROF utility is a stand-alone utility that is executed directly from the operating system prompt. The following shows a simple TKPROF formatting example taken from a Unix system:

```
$ tkprof ora_1253.trc trace.txt
```

This example would format a user trace file called `ora_1253.trc` and store the formatted version in a file called `trace.txt`. As this example shows, the first two parameters to follow the TKPROF command are the name of the trace file and the output file, respectively. The TKPROF utility has
several additional command line arguments that can also be used when formatting the trace file, depending on what information is desired. These options are specified on the command line immediately after the required trace file name and output file name parameters. A listing of these options is shown in Table 3.1.

**TABLE 3.1**  
**TKPROF Command Line Options**

<table>
<thead>
<tr>
<th>Option</th>
<th>Definition and Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXPLAIN</td>
<td>Generates an Explain Plan for each statement in the trace file by using the PLAN_TABLE in the schema specified by this command. This option must include a valid Oracle username and password.</td>
</tr>
<tr>
<td>TABLE</td>
<td>Using the schema specified by the EXPLAIN option, the execution plan is stored in the specified table rather than default table, PLAN_TABLE.</td>
</tr>
<tr>
<td>SYS</td>
<td>Indicates whether the formatted file should include recursive SQL statements (i.e., statements implicitly issued against the data dictionary in support of the primary SQL statements).</td>
</tr>
<tr>
<td>SORT</td>
<td>Determines the order in which the statements in the formatted trace file will be sorted. The 22 sort options are listed in Tables 3.2, 3.3, and 3.4.</td>
</tr>
<tr>
<td>RECORD</td>
<td>Specifies the filename into which the SQL statements in the trace file should be written. Allows you to isolate the SQL statements without any accompanying resource usage information. Recursive SQL will be omitted.</td>
</tr>
<tr>
<td>PRINT</td>
<td>Limits the number of SQL statements included in the resulting formatted trace file to the specified value.</td>
</tr>
<tr>
<td>INSERT</td>
<td>Creates an SQL script that, when executed, will create a table called TKPROF_TABLE and then populate this table with the raw statistics contained in this trace file.</td>
</tr>
</tbody>
</table>
Several examples demonstrating the use of TKPROF with some of these options are shown below.

The following demonstrates the use of the TKPROF with SYS and EXPLAIN parameters on a Unix system:

```bash
$ tkprof ora_1234.trc trace.txt sys=no explain=joe/sidney@PROD
```

This use of TKPROF would:

- Format the trace file `ora_1234.trc` and store the results in an output file called `trace.txt`.
- Cause any recursive SQL in the original trace file to be omitted from the formatted trace file, `trace.txt`.
- Include in the formatted trace file, `trace.txt`, Explain Plans of each SQL statement issued. The PLAN_TABLE in the user Joe’s schema in the PROD database would be used to generate the Explain Plans.

The following demonstrates the use of TKPROF with RECORD and PRINT parameters on a Windows 2000 system:

```bash
C:\> tkprof ora00740.trc trace.txt print=10 record=sql.txt
```

This use of the TKPROF utility would:

- Format the trace file `ora00740.trc` and store the first 10 SQL statements in an output file called `trace.txt`.
- Create a file called `sql.txt` that would contain the first ten SQL statements in the original trace file.

The following demonstrates the use of TKPROF with INSERT and SORT parameters on a Unix system:

```bash
$ tkprof ora_1702.trc trace.txt insert=trace.sql sort=(fchrow)
```

---

**TABLE 3.1** TKPROF Command Line Options (continued)

<table>
<thead>
<tr>
<th>Option</th>
<th>Definition and Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>AGGREGATE</td>
<td>Determines whether the statistics from multiple issuers of the same SQL statement will be counted together or independently.</td>
</tr>
<tr>
<td>WAITS</td>
<td>Calculates summary statistics for wait events that occurred within the trace. Wait events are discussed in Chapter 9, “Tuning Contention.”</td>
</tr>
</tbody>
</table>
This use of the TKPROF utility would:

- Format the trace file `ora_1702.trc` and store the results, sorted by the number of rows returned by each statement, in an output file called `trace.txt`. A discussion of other TKPROF sorting options appears in the next section.

- Create a second file called `trace.sql` that contains the SQL commands needed to create a table called `TKPROF_TABLE` and then populate it with the statistics for the SQL statements in the trace file.

Without a sort parameter like `FCHROW`, the TKPROF utility would list the SQL statements in the formatted output file in the order in which they were executed. When this default sort order is not desired, TKPROF has several additional sorting parameters. The formatted output will be sorted in descending order, according to the option you specify. Multiple sort parameters can also be used simultaneously. In these cases the sorted output will be sorted in the order specified, but is based on the sum of the values for the sorted option.

**TKPROF Sorting Parameters**

Each parameter is related to one of the three phases of SQL statement processing: Parse, Execute, or Fetch. Parsing refers to the process of getting an SQL statement ready to execute. This process includes checking the statement for syntactic correctness and determining the user’s privileges on objects referenced in the statement. TKPROF sort options related to the parse phase of SQL statement processing are shown in Table 3.2.

<table>
<thead>
<tr>
<th>Option</th>
<th>Sort Order</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRSCNT</td>
<td>Number of times the statement was parsed</td>
</tr>
<tr>
<td>PRSCPU</td>
<td>Amount of CPU time spent parsing the statement</td>
</tr>
<tr>
<td>PRSELA</td>
<td>Elapsed time spent parsing the statement</td>
</tr>
<tr>
<td>PRSDSK</td>
<td>Number of disk reads that occurred while the statement was being parsed</td>
</tr>
<tr>
<td>PRSMIS</td>
<td>Number of Library Cache misses while the statement was being parsed</td>
</tr>
</tbody>
</table>
The second step in SQL statement processing, *Execute*, occurs when the SQL statement performs the action specified in the statement. Examples of execution might be selecting, inserting, updating, or deleting data from a segment, or creating a user, tablespace, or datafile. Table 3.3 shows the TKPROF sort options related to the execute phase of SQL statement processing.

**TABLE 3.3** Sort Options for the Execute Phase of Statement Processing

<table>
<thead>
<tr>
<th>Option</th>
<th>Sort Order</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXECNT</td>
<td>Number of times the statement was executed</td>
</tr>
<tr>
<td>EXECPU</td>
<td>Amount of CPU time spent executing the statement</td>
</tr>
<tr>
<td>EXEELA</td>
<td>Elapsed time spent executing the statement</td>
</tr>
<tr>
<td>EXEDSK</td>
<td>Number of disk reads that occurred while the statement was being executed</td>
</tr>
<tr>
<td>EXEQRY</td>
<td>Number of rollback buffers read from the SGA while the statement was executing</td>
</tr>
<tr>
<td>EXECU</td>
<td>Number of data buffers read from the SGA while the statement was executing</td>
</tr>
<tr>
<td>EXEROW</td>
<td>Number of rows that were processed while the statement was executing</td>
</tr>
<tr>
<td>EXEMIS</td>
<td>Number of Library cache misses that occurred while the statement was executing</td>
</tr>
</tbody>
</table>
If the SQL statement that was issued was a SELECT statement, it will also perform the last step in SQL statement processing—a Fetch operation. The Fetch phase occurs when the result set returned by the SQL statement is sent back to the user. TKPROF sort options related to the fetch phase of SQL statement processing are shown in Table 3.4.

### Table 3.4 Sort Options for the Fetch Phase of Statement Processing

<table>
<thead>
<tr>
<th>Option</th>
<th>Sort Order</th>
</tr>
</thead>
<tbody>
<tr>
<td>FCHCNT</td>
<td>Number of row fetches that occurred</td>
</tr>
<tr>
<td>FCHCPU</td>
<td>Amount of CPU time spent fetching rows</td>
</tr>
<tr>
<td>FCHELA</td>
<td>Elapsed time spent fetching rows</td>
</tr>
<tr>
<td>FCHDSK</td>
<td>Number of disk reads that occurred while the rows were being fetched</td>
</tr>
<tr>
<td>FCHQRY</td>
<td>Number of rollback blocks read from the SGA while the rows were being fetched</td>
</tr>
<tr>
<td>FCHCU</td>
<td>Number of data blocks read from the SGA while the rows were being fetched</td>
</tr>
<tr>
<td>FCHROW</td>
<td>Number of rows fetched by the statement</td>
</tr>
</tbody>
</table>

In addition to the sort parameters shown in Tables 3.2, 3.3, and 3.4, TKPROF output can also be sorted by the user ID of the user who parsed the statement using the USERID sort parameter.

**Interpreting a Trace File Formatted Using TKPROF**

Once a user trace file has been formatted using the TKPROF utility, the contents of the resulting file can be used as an excellent measure of SQL performance. A sample trace file formatted with TKPROF is shown in Figure 3.1.
FIGURE 3.1 Sample output from TKPROF formatted trace file

```sql
SELECT /*+ RULE */ bucket_cnt, row_cnt, cache_cnt, null_cnt, timestmap,
    sample_size, minimum, maximum, distinct, lowval, highval, density, col,
    spand, parsex, ampcol
FROM hist_head WHERE obj# = 1 AND hmtcol# = 2
```

<table>
<thead>
<tr>
<th>call</th>
<th>count</th>
<th>cpu</th>
<th>elapsed</th>
<th>disk</th>
<th>query</th>
<th>current</th>
<th>rows</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parse</td>
<td>2</td>
<td>0.07</td>
<td>0.19</td>
<td>2</td>
<td>16</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Execute</td>
<td>11</td>
<td>0.00</td>
<td>0.00</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Fetch</td>
<td>11</td>
<td>0.00</td>
<td>0.00</td>
<td>2</td>
<td>12</td>
<td>0</td>
<td>11</td>
</tr>
<tr>
<td>total</td>
<td>14</td>
<td>0.07</td>
<td>0.11</td>
<td>4</td>
<td>18</td>
<td>1</td>
<td>11</td>
</tr>
</tbody>
</table>

MISSES in library cache during parse: 1
Optimizer goal: RULE
Parsin user id: SYS (recursive depth: 1)

```sql
SELECT employee_id, last_name, hire_date, job_id
FROM hr.employees
WHERE salary > 55000
```

<table>
<thead>
<tr>
<th>call</th>
<th>count</th>
<th>cpu</th>
<th>elapsed</th>
<th>disk</th>
<th>query</th>
<th>current</th>
<th>rows</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parse</td>
<td>3</td>
<td>0.08</td>
<td>0.18</td>
<td>0</td>
<td>146</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Execute</td>
<td>1</td>
<td>0.00</td>
<td>0.00</td>
<td>3</td>
<td>3</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>Fetch</td>
<td>2</td>
<td>0.00</td>
<td>0.00</td>
<td>3</td>
<td>3</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>total</td>
<td>6</td>
<td>0.08</td>
<td>0.22</td>
<td>3</td>
<td>112</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

MISSES in library cache during parse: 1
Optimizer goal: RULE
Parsin user id: 5

```sql
INSERT INTO hr.employees
(employee_id, last_name, email, hire_date, job_id)
VALUES
('john', 'johnson', 'johnson@hr.com', 2, 5)
```

<table>
<thead>
<tr>
<th>call</th>
<th>count</th>
<th>cpu</th>
<th>elapsed</th>
<th>disk</th>
<th>query</th>
<th>current</th>
<th>rows</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parse</td>
<td>1</td>
<td>0.01</td>
<td>0.00</td>
<td>5</td>
<td>1</td>
<td>11</td>
<td>1</td>
</tr>
<tr>
<td>Execute</td>
<td>1</td>
<td>0.01</td>
<td>0.01</td>
<td>5</td>
<td>1</td>
<td>11</td>
<td>1</td>
</tr>
<tr>
<td>Fetch</td>
<td>0</td>
<td>0.00</td>
<td>0.00</td>
<td>5</td>
<td>1</td>
<td>11</td>
<td>1</td>
</tr>
<tr>
<td>total</td>
<td>2</td>
<td>0.02</td>
<td>0.11</td>
<td>5</td>
<td>1</td>
<td>11</td>
<td>1</td>
</tr>
</tbody>
</table>

The CPU and Elapsed timings shown in the example above will only be non-zero if the init.ora parameter is TIMED_STATISTICS=True.

When you examine the above trace file, you will notice the following elements:

- The first listing is for a recursive statement that was issued against the data dictionary to support the SELECT statement that follows. Since this level of detail is rarely useful, the SYS=NO option can be used during formatting to eliminate these types of entries from the output file.
• The second listing, a SELECT statement, contains information for the Parse, Execute, and Fetch phases.

• The third listing, an INSERT statement, does not return any rows (e.g., Fetch) so it contains information for only the parse and execute phases.

Table 3.5 shows the descriptions of each of the columns in the formatted TKPROF output file.

<table>
<thead>
<tr>
<th>Statistic Name</th>
<th>Statistic Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Count</td>
<td>The number of times the Parse, Execute, or Fetch phase occurred</td>
</tr>
<tr>
<td>CPU</td>
<td>Total number of seconds the CPU spent processing all the Parse, Execute, or Fetch calls in the statement</td>
</tr>
<tr>
<td>Elapsed</td>
<td>Total number of seconds spent processing all the Parse, Execute, or Fetch calls in the statement</td>
</tr>
<tr>
<td>Disk</td>
<td>Total number of data blocks read from disk during the Parse, Execute, and Fetch calls in the statement</td>
</tr>
<tr>
<td>Query</td>
<td>Total number of rollback blocks (blocks in consistent mode) read from the SGA during the Parse, Execute, and Fetch calls in the statement</td>
</tr>
<tr>
<td>Current</td>
<td>Total number of data blocks (blocks in current mode) read from the SGA during the Parse, Execute, and Fetch calls in the statement</td>
</tr>
<tr>
<td>Rows</td>
<td>The number of rows affected by the Execute phase for INSERT, UPDATE, and DELETE statements, or Fetch phase for SELECT statements</td>
</tr>
</tbody>
</table>

Using these statistics, you can see that the second statement was parsed once, executed once, and performed two fetches. This processing consumed a total of .06 seconds of CPU time, and .22 seconds elapsed while the statement was processed. The statement required that 3 data blocks be read from disk and $152 + 2 = 154$ from the SGA, and returned 3 rows. The
TKPROF utility will also display these statistics in summary form at the end of the output file. These statistics will provide overall totals for both recursive and non-recursive statements.

**Identifying Statements That May Require Tuning**

Using the formatted output from TKPROF, it is relatively easy to zero in on SQL statements that may require further tuning. In general you would look for statements that contain one or more of the following shortcomings:

- Consume excess CPU resources
- Take a long time to Parse, Execute, or Fetch
- Read too many data blocks from disk and too few from the SGA
- Access many data blocks but return only a few rows

As an alternative to the TKPROF utility, the Top SQL component of the OEM Diagnostics Pack can also be used to identify SQL statements that exhibit the above shortcomings. Figure 3.2 displays a sample of the output from the Top SQL GUI tool.

**FIGURE 3.2** Sample output from Top SQL
The output shown in Figure 3.2 indicates that the most resource-intensive SQL statement issued during the Top SQL monitoring period was the update on the JOB_HISTORY table, which modified the value for the column START_DATE. As evidenced by the values in the first, second, and third columns in the Top SQL output, this update generated 416 Disk Reads Per Execution and 84,801 Buffer Gets Per Execution in only one execution of the statement itself.

Once you have used either TKPROF or Top SQL to identify the SQL statements that meet the above criteria for tuning, you can use the Explain Plan facility to further determine why these statements are performing poorly.

The Explain Plan Facility

Oracle’s Explain Plan facility is used to determine how a particular SQL statement is processing. With this information, it is possible to improve the statement’s performance by rewriting the SQL code to eliminate unwanted behavior.

While a complete discussion of interpreting Explain Plan output is beyond the scope of this text and the Oracle9i Performance Tuning OCP exam, you should have a basic understanding of this important tool.

Explain Plans can be generated using the TKPROF utility or using the EXPLAIN PLAN FOR command. Each of these techniques is discussed in the following sections.

Generating Explain Plans Using TKPROF

As you saw in the previous section, the EXPLAIN parameter can be used to generate Explain Plans for SQL statements when formatting a user trace file with the TKPROF utility. Figure 3.3 shows what this Explain Plan output would look like.

However, formatting a large trace file using the EXPLAIN parameter can take a prohibitively long time. In these cases, you will probably want to examine the formatted trace file for poorly performing SQL statements and then generate Explain Plans for only those statements using the techniques discussed in the next section.
FIGURE 3.3 Explain Plan output generated by TKPROF

```sql
select l.city, d.department_name, e.first_name, e.last_name
from hr.departments d, hr.employees e, hr.locations l
where d.department_id = e.department_id
and d.location_id = l.location_id
and e.department_name = 'Shipping';
```

<table>
<thead>
<tr>
<th>call</th>
<th>count</th>
<th>cpu</th>
<th>elapsed</th>
<th>disk</th>
<th>query</th>
<th>current</th>
<th>row</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parse</td>
<td>1</td>
<td>0.02</td>
<td>0.02</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Execute</td>
<td>1</td>
<td>0.00</td>
<td>0.00</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Fetch</td>
<td>4</td>
<td>0.00</td>
<td>0.00</td>
<td>1.3</td>
<td>2</td>
<td>2</td>
<td>45</td>
</tr>
<tr>
<td>total</td>
<td>6</td>
<td>0.02</td>
<td>0.03</td>
<td>1.3</td>
<td>2</td>
<td>2</td>
<td>45</td>
</tr>
</tbody>
</table>

Misses in library cache during parse: 0
Optimizer goal: CHOOSE
Parsing user id: 5 (SYSTEM)

ROWS: Row Source Operation
45 TABLE ACCESS BY INDEX ROWID EMPLOYEES
47 NESTED LOOPS
31 TABLE ACCESS FULL DEPARTMENTS
31 TABLE ACCESS BY INDEX ROWID LOCATIONS
45 INDEX RANGE SCAN (object id 3279)

ROWS: Execution Plan
45 SELECT STATEMENT GOAL: CHOOSE
47 NESTED LOOPS
31 NESTED LOOPS
31 TABLE ACCESS GOAL: ANALYZED (FULL) OF 'DEPARTMENTS'
31 TABLE ACCESS GOAL: ANALYZED (INDEX ROWID) OF 'LOCATIONS'
31 INDEX GOAL: ANALYZED (INDEX SCAN) OF 'DEPT_PK'
45 INDEX GOAL: ANALYZED (INDEX RANGE SCAN) OF 'EMPLOYEE_PK'

Generating Explain Plans Using `EXPLAIN PLAN FOR`

Once a poorly performing SQL statement has been identified, you can use the `EXPLAIN PLAN FOR` command to generate an Explain Plan for this statement. These are the steps to generate an Explain Plan using this technique:

- Create a Plan_Table table using the utlxplan.sql script found in `$ORACLE_HOME/rdbms/admin` on Unix systems or `%ORACLE_HOME%\rdbms\admin\` on Windows 2000 systems. This step only needs to be performed once. Here is an example of the script being executed on a Unix system:

  ```sql
  SQL> @$ORACLE_HOME/rdbms/admin/utlxplan.sql
  ```
• Populate the PLAN_TABLE with the SQL statement’s execution plan using the EXPLAIN PLAN FOR command. This is demonstrated here using a query on three tables called DISTRIBUTOR, DISTRICT, and EMPLOYEE:

```sql
SQL> EXPLAIN PLAN FOR
2  SELECT dist.distributor_id,
3       dist.city,
4       dist.state,
5       dist.zip_code,
6       district.name,
7       emp.last_name
8 FROM   distributor dist, district, employee emp
9 WHERE  emp.employee_id = dist.manager_id
10 AND    district.district_id = dist.district_id;
```

• The EXPLAIN PLAN FOR command will populate the PLAN_TABLE with the execution plan for the query. The query itself does not execute. Only its execution plan is generated.

---

**Tip**

Be sure to either truncate the PLAN_TABLE or use the SET STATEMENT_ID= syntax between executions of the EXPLAIN PLAN FOR command. Otherwise, the execution plans from several queries will be stored in the PLAN_TABLE at the same time, making interpretation difficult.

• You now can query the PLAN_TABLE in order to see how the explained statement would be executed. The following query can be used to extract the important execution plan information from the PLAN_TABLE:

```sql
SQL> SELECT LPAD(' ',4*(LEVEL-2))||
2               operation||' '||
3               options||' '||
4               object_name "EXECUTION_PLAN"
5  FROM plan_table
6  START WITH id = 0
7  CONNECT BY PRIOR id = parent_id;
```
The output from this query on the PLAN_TABLE looks like the following sample:

EXECUTION_PLAN
----------------------------------------------
SELECT STATEMENT
NESTED LOOPS
NESTED LOOPS
  TABLE ACCESS FULL DISTRIBUTOR
  TABLE ACCESS BY INDEX ROWID EMPLOYEE
  INDEX UNIQUE SCAN EMPLOYEE_ID_PK
  TABLE ACCESS BY INDEX ROWID DISTRICT
  INDEX UNIQUE SCAN DISTRICT_ID_PK

Whether generated by TKPROF or EXPLAIN PLAN FOR, you will need to examine the output from the PLAN_TABLE to identify if any of the individual operations in the SQL processing can benefit from tuning.

Oracle also provides two built-in SQL scripts for querying the content of the PLAN_TABLE, which can be used in place of the sample query shown above. Both of these scripts are located in $ORACLE_HOME/rdbms/admin on Unix systems and %ORACLE_HOME%\rdbms\admin on Windows 2000 systems. The first script, utlxpls.sql excludes any Parallel Query information in the Explain Plan. The second script, utlxplp.sql, includes Parallel Query information in the Explain Plan.

Interpreting Explain Plan Output

In general, Explain Plan output is interpreted by starting at the innermost operation in the Explain Plan. This operation is always executed first, unless it is an index access. In this case the table operation directly above the index access would begin the execution plan. From this starting point, you examine the Explain Plan’s remaining operations, working your way up and out toward the leftmost operations in the plan.

If two operations appear at the same level and are both the innermost operations, the operation on top is executed first.
Gaining proficiency at reading Explain Plans takes practice. In order to introduce you to this process, the output from the previous example is interpreted here:

- The innermost operation of this plan is INDEX UNIQUE SCAN EMPLOYEE_ID_PK.
- Because this step involves an index and not a table, the operation directly above this step, TABLE ACCESS BY INDEX ROWID EMPLOYEE, is instead considered as the possible start to this execution plan. You do this because the index lookup and the table access are essentially one step, not two.
- However, at the same level as the access of EMPLOYEE, is the operation TABLE ACCESS FULL DISTRIBUTOR. Since these two operations appear at the same level, the operation on top, TABLE ACCESS FULL DISTRIBUTOR, is where the execution plan begins. Note that while the operation INDEX UNIQUE SCAN DISTRICT_ID_PK is also on the same level as the access of EMPLOYEE and DISTRIBUTOR, this index access is ignored at this time because it involves a table that is on another, more outer level.

A TABLE ACCESS FULL operation in an Explain Plan indicates that a Full Table Scan (FTS) occurred. These operations cause every row in a table to be accessed. This can be a costly operation depending on the number of rows in a table and the proportion of those rows returned by the query. The appropriate use of indexing helps minimize performance problems associated with FTS. Indexing is discussed later in this chapter.

- Working your way out and up, the next operation you encounter is NESTED LOOPS. In this operation, the DISTRIBUTOR table is joined to the EMPLOYEE table via the EMPLOYEE_ID_PK index. A NESTED LOOPS join occurs whenever Oracle takes the resulting rows from the first operation (the FTS of DISTRIBUTOR) and then compares the results of that operation to each row in the second operation (access of EMPLOYEE via an index).

To better understand what has occurred up to this point in the execution plan, look at the example below, which has numbered each operation.
in the Explain Plan in the order it occurs:

<table>
<thead>
<tr>
<th>Order</th>
<th>Execution Plan</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SELECT STATEMENT</td>
</tr>
<tr>
<td></td>
<td>NESTED LOOPS</td>
</tr>
<tr>
<td>4</td>
<td>NESTED LOOPS</td>
</tr>
<tr>
<td>1</td>
<td>TABLE ACCESS FULL DISTRIBUTOR</td>
</tr>
<tr>
<td>3</td>
<td>TABLE ACCESS BY INDEX ROWID EMPLOYEE</td>
</tr>
<tr>
<td>2</td>
<td>INDEX UNIQUE SCAN EMPLOYEE_ID_PK</td>
</tr>
<tr>
<td>6</td>
<td>TABLE ACCESS BY INDEX ROWID DISTRICT</td>
</tr>
<tr>
<td>5</td>
<td>INDEX UNIQUE SCAN DISTRICT_ID_PK</td>
</tr>
</tbody>
</table>

Now consider the interpretation:

- Since the NESTED LOOPS join and TABLE ACCESS BY INDEX ROWID DISTRIBUTOR and EMPLOYEE, both appear at this level, we consider them next. These operations imply that the results of the first operation, a NESTED LOOPS join of DISTRIBUTOR and EMPLOYEE, is itself joined to DISTRICT via the DISTRICT_ID_PK index using a second NESTED LOOPS join found on the next level.

- The only operation left is to return the rows to the user. This step is not expressed in the Explain Plan, but is implied by the SELECT STATEMENT keyword that appears at the beginning of the Explain Plan output.

To conclude the previous example, the final stages of the Explain Plan are numbered in the order in which they occur:

<table>
<thead>
<tr>
<th>Order</th>
<th>Execution Plan</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>SELECT STATEMENT</td>
</tr>
<tr>
<td>7</td>
<td>NESTED LOOPS</td>
</tr>
<tr>
<td>4</td>
<td>NESTED LOOPS</td>
</tr>
<tr>
<td>1</td>
<td>TABLE ACCESS FULL DISTRIBUTOR</td>
</tr>
<tr>
<td>3</td>
<td>TABLE ACCESS BY INDEX ROWID EMPLOYEE</td>
</tr>
<tr>
<td>2</td>
<td>INDEX UNIQUE SCAN EMPLOYEE_ID_PK</td>
</tr>
<tr>
<td>6</td>
<td>TABLE ACCESS BY INDEX ROWID DISTRICT</td>
</tr>
<tr>
<td>5</td>
<td>INDEX UNIQUE SCAN DISTRICT_ID_PK</td>
</tr>
</tbody>
</table>
The AUTOTRACE Utility

The Oracle Autotrace utility combines elements of both the TKPROF utility and the EXPLAIN PLAN FOR facility. Unlike TKPROF, AUTOTRACE does not require you to format a trace file to get resource usage information. This information is automatically included in the AUTOTRACE output along with the execution plan of the SQL statement that was issued. No separate query on the PLAN_TABLE is required. However, unlike the EXPLAIN PLAN FOR facility, AUTOTRACE actually executes the statement being examined before generating an execution plan.

Some preliminary configuration operations must be performed before the AUTOTRACE utility can be utilized to examine SQL execution paths. These requisite configuration steps and subsequent techniques for using the AUTOTRACE utility are described in the following sections.

Preparing to Use AUTOTRACE

Before using the AUTOTRACE utility, you must first perform several initial setup operations. These operations are performed only once.

First, each user who wishes to use the AUTOTRACE utility will need a PLAN_TABLE table in their schema. This table can be created using the utlxplan.sql script found in $ORACLE_HOME/rdbms/admin on Unix systems or %ORACLE_HOME%\rdbms\admin on Windows 2000 systems. Unlike EXPLAIN PLAN FOR, AUTOTRACE will automatically clean out the contents of the PLAN_TABLE between executions.

Second, you will then need to create a database role called PLUSTRACE by running an Oracle-supplied script called plustrce.sql while logged on as the user SYS. This script is located in $ORACLE_HOME/sqlplus/admin on Unix systems or %ORACLE_HOME%\sqlplus\admin on Windows 2000 systems.

Finally, you will have to grant the new PLUSTRACE role to every user who will be using the AUTOTRACE utility.

Using AUTOTRACE

The AUTOTRACE utility is only available from within a SQL*Plus session. AUTOTRACE can be activated in a SQL*Plus session by issuing the SET AUTOTRACE ON command. The following example demonstrates the output of the AUTOTRACE utility:

```sql
SQL> set autotrace on
SQL> SELECT dist.distributor_id 'ID',
   2     dist.city,
   3```

Measuring SQL Performance 103

```sql
3 dist.state,
4 dist.zip_code,
5 district.name,
6 emp.last_name
7 FROM distributor dist,
8        district,
9        employee emp
10 WHERE emp.employee_id = dist.manager_id
11 AND district.district_id = dist.district_id;
```

<table>
<thead>
<tr>
<th>ID</th>
<th>CITY</th>
<th>STATE</th>
<th>ZIP_CODE</th>
<th>NAME</th>
<th>LAST_NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>Madison</td>
<td>WI</td>
<td>53704</td>
<td>Northern</td>
<td>Bryla</td>
</tr>
<tr>
<td>110</td>
<td>Freeport</td>
<td>IL</td>
<td>61032</td>
<td>Midwest</td>
<td>Gastel</td>
</tr>
<tr>
<td>205</td>
<td>Phoenix</td>
<td>AZ</td>
<td>85024</td>
<td>Southwest</td>
<td>Girardin</td>
</tr>
<tr>
<td>355</td>
<td>El Cajon</td>
<td>CA</td>
<td>92020</td>
<td>Western</td>
<td>Anderson</td>
</tr>
<tr>
<td>451</td>
<td>Boston</td>
<td>MA</td>
<td>02152</td>
<td>Eastern</td>
<td>Wojtkunski</td>
</tr>
</tbody>
</table>

Execution Plan

<table>
<thead>
<tr>
<th>0 SELECT STATEMENT Optimizer=CHOOSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 NESTED LOOPS</td>
</tr>
<tr>
<td>2 1 NESTED LOOPS</td>
</tr>
<tr>
<td>3 2 TABLE ACCESS (FULL) OF 'S_WAREHOUSE'</td>
</tr>
<tr>
<td>4 2 TABLE ACCESS (BY INDEX ROWID) OF 'S_EMP'</td>
</tr>
<tr>
<td>5 4 INDEX (UNIQUE SCAN) OF 'S_EMP_ID_PK' (UNIQUE)</td>
</tr>
<tr>
<td>6 1 TABLE ACCESS (BY INDEX ROWID) OF 'S_REGION'</td>
</tr>
<tr>
<td>7 6 INDEX (UNIQUE SCAN) OF 'S_REGION_ID_PK' (UNIQUE)</td>
</tr>
</tbody>
</table>

Statistics

<table>
<thead>
<tr>
<th>0 recursive calls</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 db block gets</td>
</tr>
<tr>
<td>24 consistent gets</td>
</tr>
<tr>
<td>0 physical reads</td>
</tr>
<tr>
<td>0 redo size</td>
</tr>
<tr>
<td>1717 bytes sent via SQL*Net to client</td>
</tr>
</tbody>
</table>
786 bytes received via SQL*Net from client
4 SQL*Net roundtrips to/from client
1 sorts (memory)
0 sorts (disk)
5 rows processed

The statistics listed at the end of the AUTOTRACE output will be explained in more detail in Chapter 8, “Tuning Disk I/O.”

In addition to ON, the SET AUTOTRACE command also has the four additional options shown in Table 3.6.

**TABLE 3.6 Options for AUTOTRACE Utility**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ON</td>
<td>Displays query results, execution plan, and statistics</td>
</tr>
<tr>
<td>ON STATISTICS</td>
<td>Displays the query results and statistics, but no execution plan</td>
</tr>
<tr>
<td>ON EXPLAIN</td>
<td>Displays query results and execution plan, but no statistics</td>
</tr>
<tr>
<td>TRACEONLY</td>
<td>Displays the execution plan and the statistics, but no query results*</td>
</tr>
<tr>
<td>TRACEONLY STATISTICS</td>
<td>Displays the statistics only, no execution plan or query results*</td>
</tr>
<tr>
<td>OFF</td>
<td>Turns off the AUTOTRACE utility</td>
</tr>
</tbody>
</table>

*The query is still executed even though the query results are not shown.

**SQL Tuning Information in STATSPACK Output**

In addition to the TKPROF, EXPLAIN PLAN, and AUTOTRACE utilities shown in the previous sections, the output from a STATSPACK report also contains
useful SQL statement tuning information. This information is found in four sections of the STATSPACK output file:

- SQL ordered by Gets
- SQL ordered by Reads
- SQL ordered by Executions
- SQL Executions ordered by Parse Calls

**SQL Ordered by Gets**

The first section of the STATSPACK output related to SQL statement tuning is shown in Figure 3.4.

![Figure 3.4 SQL ordered by Gets STATSPACK output](image)

Whenever SQL statements are issued, Oracle checks to see if a parsed version of that SQL statement already exists in the SGA. Finding a cached version of a SQL statement in the SGA is referred to as a *Get*. This section of the STATSPACK output shows the SQL statements that had the highest number of Gets during the time between the two snapshot periods. Improving the performance of Gets for application SQL is covered in Chapter 4, “Tuning the Shared Pool.”

A SQL statement must have experienced at least 10,000 Gets in order to be included in the STATSPACK output.
SQL Ordered by Reads

The second section of the STATSPACK output related to SQL statement tuning is shown in Figure 3.5.

**Figure 3.5** SQL ordered by Reads STATSPACK output

This section of the STATSPACK output shows the SQL statements that caused the highest amount of read activity during the time between the two snapshot periods. Improving the performance of Read activity is discussed in Chapter 5, “Tuning the Database Buffer Cache,” and Chapter 8, “Tuning Disk I/O.”

A SQL statement must have performed at least 1,000 Reads in order to be included in the STATSPAK output.

**SQL Ordered by Executions**

The third section of the STATSPACK output related to SQL statement tuning is shown in Figure 3.6.

This section of the STATSPACK output shows the SQL statements that were most frequently executed during the time between the two snapshot periods. Knowing which statements are frequently executed can be useful when deciding which PL/SQL packages and procedures to pin in memory. Pinning of PL/SQL packages is covered in Chapter 4, “Tuning the Shared Pool.”
A SQL statement must have been executed at least 100 times in order to be included in theSTATSPACK output.

**SQL Executions Ordered by Parse Calls**

The last section of the STATSPACK output related to SQL statement tuning is shown in Figure 3.7.

**FIGURE 3.7** SQL ordered by Parse calls STATSPACK output
This section of the STATSPACK output shows the SQL statements that were most frequently parsed during the time between the two snapshot periods. Knowing which statements are frequently being parsed can help you identify statements that could be potentially re-written to improve SQL reuse in the Shared Pool. This concept is discussed in Chapter 4, “Tuning the Shared Pool.”

A SQL statement must have received at least 1,000 Parse calls in order to be included in the STATSPACK output.

Each of the threshold values that determine whether an SQL statement qualifies for inclusion in the STATSPACK output can be changed to suit your needs. These new threshold values can be set either temporarily or permanently. See Chapter 21, “Using STATSPACK,” in the Oracle9i Database Performance Guide and Reference Release 1 (9.0.1) (Part Number A87503-02) for details on how to change these thresholds.

Understanding the Oracle Optimizer

As you learned in the previous section, several tools exist that will allow you to see a particular SQL statement’s execution plan. A single SQL statement often has many different possible execution plans, all of which return the same result, but only one of which is optimal. It is the job of the Oracle optimizer to choose the best execution plan.

<table>
<thead>
<tr>
<th><strong>Objective</strong></th>
<th>Describe how the Optimizer is used</th>
</tr>
</thead>
</table>

There are essentially two optimizer modes in Oracle: Rule and Cost. As you will see in the following sections, these two optimizer modes differ in
terms of how they select the optimal execution plan from among the many possible execution plans for a given statement.

**Rule-Based Optimization**

The rule-based optimizer (RBO) uses a set of predefined “rules of thumb” to decide how to best execute a query. These guidelines include rules like, “If a table column referenced in a WHERE clause has an index on it, always use the index when accessing the corresponding table data.” In early versions of Oracle, such as Version 6 and older, application developers frequently used their knowledge of these rules to directly influence the behavior of the RBO.

However, rules such as this may make the RBO inefficiently utilize indexes even on very small tables that contain only a few rows. Such behavior requires Oracle to perform two I/Os (one on the index and one on the table) when the entire table could just as easily been accessed with only one I/O by skipping the use of the index.

In addition to table size, column cardinality is also ignored by the RBO when making execution plan determinations. Cardinality refers to the variation in the data within a column. Columns that contain many distinct values are said to have high cardinality. Columns that contain very few distinct values are said to have low cardinality. At times this inability to consider cardinality leads the RBO to follow ineffective SQL execution plans by utilizing indexes even when the majority of the table’s rows are ultimately going to be read and retrieved by the query. As with the previous example, this scenario also leads to unnecessary I/O.

While the RBO is still supported for backward compatibility purposes, Oracle now recommends that the cost-based optimizer be used in favor of the RBO in all new development environments.

**Cost-Based Optimization**

Unlike the rule-based optimizer that has only its predefined guidelines to follow when executing a query, the cost-based optimizer (CBO) considers many different execution plans and then selects the one with the lowest execution cost. While previous versions of Oracle primarily considered logical I/O to be the primary determinant of a SQL statement’s cost of
execution, Oracle9i also weighs the cost of CPU and sorting when determining a SQL statement’s overall cost. The following simple example shows how these three CBO cost factors can be included in an Explain Plan query:

```
SQL> EXPLAIN PLAN FOR
  2  SELECT e.employee_id, d.department_name
  3  FROM employees e, departments d
  4  WHERE e.department_id=d.department_id;

Explained.

SQL> SELECT LPAD(' ',4*(LEVEL-2))||
  2               operation||' '||
  3               options||' '||
  4               object_name 'EXECUTION_PLAN',
  5       io_cost,
  6       cpu_cost,
  7       temp_space
  8  FROM plan_table
  9  START WITH id = 0
 10  CONNECT BY PRIOR id = parent_id;
```

<table>
<thead>
<tr>
<th>EXECUTION_PLAN</th>
<th>IO_COST</th>
<th>CPU_COST</th>
<th>TEMP_SPACE</th>
</tr>
</thead>
<tbody>
<tr>
<td>SELECT STATEMENT</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>HASH JOIN</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TABLE ACCESS FULL DEPARTMENTS</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TABLE ACCESS FULL EMPLOYEES</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

The values associated with each of the CBO cost components are described in Table 3.7.

Null values for the IO_COST or CPU_COST columns in the Explain Plan output indicate that the rule-based optimizer was used to evaluate that portion of the statement.
Null values for the TEMP_SPACE column in the Explain Plan output indicates that the rule-based optimizer was used to evaluate that portion of the statement, or that no temporary space was needed.

<table>
<thead>
<tr>
<th><strong>Column</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>IO_COST</td>
<td>The estimated cost, in terms of I/O, associated with that operation in the query’s execution plan</td>
</tr>
<tr>
<td>CPU_COST</td>
<td>The estimated cost, in terms of CPU cycles, associated with that operation in the query’s execution plan</td>
</tr>
<tr>
<td>TEMP_SPACE</td>
<td>The estimated temporary space, in bytes, associated with that operation in the query’s execution plan</td>
</tr>
</tbody>
</table>

In order to determine the relative cost of each execution plan, the CBO relies heavily on the presence of statistics about the tables and indexes in the database. These statistics are stored in the data dictionary, which includes the following:

- The size of each table or index
- The number of rows each table or index contains
- The number of database blocks used by each table or index
- The length of each table row
- The cardinality of the column data in indexed columns

By default, these statistics are not present in the database. The statistics are gathered and stored in the database when the tables and indexes are analyzed. Database statistics can be gathered at the individual table and index level, at the schema level, or for the entire database. Techniques for gathering statistics are discussed in the following sections.
The following command demonstrates how to analyze an individual table and index in order to gather its statistics:

```
SQL> ANALYZE INDEX employee_last_name_idx COMPUTE STATISTICS;
SQL> ANALYZE TABLE employee COMPUTE STATISTICS;
```

The first command gathers the statistics for an index called EMPLOYEE_LAST_NAME_IDX. The second command gathers the statistics for the EMPLOYEE table by examining every row in the table. It will also gather statistics on all the indexes associated with the EMPLOYEE table. If EMPLOYEE is a very large table, this method could be time consuming and resource intensive. Larger tables can be analyzed using this command:

```
SQL> ANALYZE INDEX distributor_id_pk ESTIMATE STATISTICS;
```

Using ESTIMATE in place of COMPUTE causes Oracle to estimate table and index statistics using a sample of the table or index data. The default sample size is 1,064 rows. In most cases, the ESTIMATE parameter provides statistics that are nearly as accurate as those collected using the COMPUTE option. You can specify your own sample size in terms of rows or a percentage of the overall table using the following commands:

```
SQL> ANALYZE TABLE distributors ESTIMATE STATISTICS
       2  SAMPLE 500 ROWS;
SQL> ANALYZE INDEX distributor_id_pk ESTIMATE STATISTICS
       2  SAMPLE 35 PERCENT;
```
If you specify a value for SAMPLE that exceeds 50 percent of the rows in the table, Oracle will generate the statistics using a COMPUTE instead of ESTIMATE.

If you desire an even more focused analysis of the data in the tables and indexes, you can include the FOR clause with the ANALYZE TABLE command:

```
SQL> ANALYZE TABLE distributors ESTIMATE STATISTICS
  2 FOR COLUMNS district_id SIZE 200;
```

This command would limit the analysis of the table to the just the DISTRICT_ID column instead of the default behavior of examining all columns. The SIZE parameter specifies how many slices to divide the column’s values into before examining the cardinality of data within each slice. The more slices that are taken, the more accurate a picture the CBO will have regarding the distribution of the data within the column. The default value for SIZE is 75. The acceptable range of values is 1 through 254. Without this information, the CBO assumes that the data is normally distributed when, in fact, it may not be. Additional options for the FOR clause of the ANALYZE command appear in Table 3.8.

### Table 3.8 Options of the FOR clause of the ANALYZE command

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FOR TABLE</td>
<td>Gathers only table statistics, without column statistics</td>
</tr>
<tr>
<td>FOR COLUMNS</td>
<td>Gathers column statistics for only the specified columns</td>
</tr>
<tr>
<td>FOR ALL COLUMNS</td>
<td>Gathers columns statistics only, on all columns</td>
</tr>
<tr>
<td>FOR ALL INDEXES</td>
<td>Gathers the statistics for all indexes on this table, but no table statistics</td>
</tr>
<tr>
<td>FOR ALL INDEXED</td>
<td>Gathers columns statistics on only columns that are indexed</td>
</tr>
<tr>
<td>COLUMNS</td>
<td>Gathers column statistics on all local indexes of a partitioned table</td>
</tr>
</tbody>
</table>
The ANALYZE command also has a DELETE option that can be used to remove statistics if needed:

```
SQL> ANALYZE TABLE employee DELETE STATISTICS;
```

**Histograms**

When a table is analyzed, Oracle stores the lowest and highest data values in each column as part of the table’s statistics.

### Oracle Objective

Describe the use of histograms

The Oracle optimizer assumes that the data between these two data values is normally distributed. In other words, the optimizer assumes that most of the data in the column falls between these high and low values, and less of the data in the column will be found at either of the two extremes. If this assumption is not true, the optimizer may make very poor choices about how to execute SQL statements that involve that column. *Histogram* statistics can be used to improve the performance of SQL statements whose underlying tables have nonuniformly distributed data.

### Real World Scenario

**The Importance of Histograms**

On one occasion, I experienced a query that was running against a large table whose indexed columns did not have normally distributed data. A particular query on this table was taking over 17 hours to parse, execute, and fetch its rows. After we realized that the data, instead of being normally distributed between a low and high value, had spikes of high and low values throughout, we analyzed the table using the following options:

```
SQL> ANALYZE TABLE finaid COMPUTE STATISTICS FOR COLUMNS award SIZE 100;
```

This command created a histogram that divided the data in the AWARD column of the FINAID table into 100 separate slices. The histogram then determined the high and low value for each of the slices and stored those as part of the table and index statistics. These statistics gave the optimizer
Analyzing by Schema

In order for the CBO to perform best, all the tables and indexes involved in an SQL statement should be analyzed. Oracle provides two PL/SQL packages, DBMS_UTILITY and DBMS_STATS, which can be used to analyze every table and index in a user’s schema without having to individually specify each segment by name. Oracle recommends the use of the DBMS_STATS package in Oracle9i over the older DBMS_UTILITY. The Oracle Enterprise Manager Console can also be used to gather table and index statistics.

**DBMS_UTILITY**

The DBMS_UTILITY package accepts several arguments that are similar to those you saw for the ANALYZE command in the previous section. An example using several of these arguments is shown here:

```sql
SQL> EXECUTE DBMS_UTILITY.ANALYZE_SCHEMA('SCOTT',
                 'ESTIMATE',0,40,'FOR ALL INDEXED COLUMNS SIZE 40');
```

Each of the arguments used in this example is described in Table 3.9.

<table>
<thead>
<tr>
<th>Argument Used in Example</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCOTT</td>
<td>The schema to be analyzed.</td>
</tr>
<tr>
<td>ESTIMATE</td>
<td>The type of analysis to be done. If COMPUTE is used, the next two arguments have no effect.</td>
</tr>
<tr>
<td>0</td>
<td>The number of rows to use in the estimate. If the estimate will be specified as a percentage of rows, enter a zero here.</td>
</tr>
</tbody>
</table>
The DBMS_UTILITY package can also be used to analyze the entire database. A database level analysis can use the same arguments that were discussed in Table 3.9, with the exception of the schema argument. The following is an example of a complete database analysis using the same sample size as the previous example:

```sql
SQL> EXECUTE SYS.DBMS_UTILITY.ANALYZE_DATABASE ('ESTIMATE', 0, 40, 'FOR ALL INDEXED COLUMNS SIZE 40');
```

Because the recursive data dictionary queries rely heavily on specific access paths to run efficiently, you should never analyze the segments in the SYS schema. Using the `ANALYZE_DATABASE` procedure will cause statistics to be gathered for the SYS schema. You should use the `DBMS_UTILITY.ANALYZE_SCHEMA('SYS', 'DELETE');` command to remove them.

### Oracle Objective

**Explain the use of the DBMS_STATS procedure**

### DBMS_STATS

The DBMS_STATS package offers several additional analyzing options that are not available in the DBMS_UTILITY package. Some of these options include the ability to:

- Back up old statistics before new statistics are gathered. This very useful feature allows you to restore some or all of the original statistics if the CBO performs poorly after updated statistics have been gathered.
- Take a random sample of *blocks*, not just rows, when estimating statistics.
Gather table statistics much faster by performing the analysis in parallel.

Automatically gather statistics on highly volatile tables and bypass gathering statistics on static tables.

The following example shows how the DBMS_STATS packages would be used to gather statistics on the EMPLOYEE table in Joe’s schema:

```sql
SQL> EXECUTE DBMS_STATS.GATHER_TABLE_STATS ('JOE', 'EMPLOYEE');
```

Like the DBMS_UTILITY package, the DBMS_STATS package can also be used to analyze indexes, an entire schema, or the whole database using the procedures shown in Table 3.10.

**TABLE 3.10** Procedures within the DBMS_STATS Package

<table>
<thead>
<tr>
<th>Procedure Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GATHER_INDEX_STATS</td>
<td>Used to gather statistics on a specified index</td>
</tr>
<tr>
<td>GATHER_SCHEMA_STATS</td>
<td>Used to gather statistics on a specified schema</td>
</tr>
<tr>
<td>GATHER_DATABASE_STATS</td>
<td>Used to gather statistics on an entire database</td>
</tr>
<tr>
<td>GATHER_SYSTEM_STATS</td>
<td>Used to gather I/O and CPU performance statistics for the server</td>
</tr>
</tbody>
</table>

Two of the procedures shown in Table 3.10, GATHER_SCHEMA_STATS and GATHER_SYSTEM_STATS, are particularly useful when utilizing the CBO.

**USING THE GATHER_SCHEMA_STATS PROCEDURE**

The GATHER_SCHEMA_STATS procedure listed in Table 3.10 is particularly useful for maintaining accurate statistics for the CBO. Using the GATHER option, you can specify one of the three following options for gathering schema statistics:

- **GATHER_STALE Option** Any table in the schema that has been designated as “monitored” using the ALTER TABLE ... MONITORING command will have its statistics gathered.
GATHER EMPTY Option Any table in the schema that has not been previously analyzed will have its statistics gathered.

GATHER AUTO Option Based on application user activity, Oracle automatically determines which tables need to be analyzed and gathers the statistics for only those tables, using the analysis method that is most appropriate.

If histogram statistics are required, Oracle9i introduced a new DBMS_STATS parameter constant called AUTO_SAMPLE_SIZE that further simplifies the management of histogram statistics. When used with the DBMS_STATS.GATHER_SCHEMA_STATS procedure, the AUTO_SAMPLE_SIZE constant will automatically determine both the proper sampling percentage to use for each segment in the schema and whether histogram statistics on those segments is warranted. The following example shows how AUTO_SAMPLE_SIZE can be used to gather statistics on Joe’s schema:

```
SQL> EXECUTE DBMS_STATS.GATHER_SCHEMA_STATS( -
  2  ownname => 'JOE', -
  3  estimate_percent => DBMS_STATS.AUTO_SAMPLE_SIZE, -
  4  method_opt => 'for all columns size AUTO' -
  5  );
```

The above command would cause Oracle to collect statistics on every table, column, and index in Joe’s schema, using sample sizes determined by Oracle. Because the AUTO option was specified in METHOD_OPT, histogram statistics would also be gathered on these segments if Oracle determines that the histograms are needed based on how frequently, and in what manner, the column data is accessed by the application. Two other options for the METHOD_OPT parameter are REPEAT and SKEWONLY. The REPEAT option will cause the DBMS_STATS package to simply repopulate any existing histogram statistics using the same number of histogram buckets as the original histogram. The SKEWONLY option will generate histogram statistics anytime a column’s data is skewed—regardless of whether the application accesses the column data in a way that will benefit from the presence of a histogram.

USING THE GATHER_SYSTEM_STATS PROCEDURE
In order for the CBO to make good choices regarding possible execution plans, it is important that the CBO be aware of the overall CPU and I/O utilization patterns on the server where the Oracle database resides. For example, if system statistics indicate that the server is generally I/O-bound rather than CPU-bound, then the CBO will select a less I/O intensive
execution plan whenever comparable I/O intensive and CPU intensive execution plans are available. The following example shows how the GATHER_SYSTEM_STATS procedure can be used to gather system statistics:

```sql
SQL> EXECUTE DBMS_STATS.CREATE_STAT_TABLE ( -
>     ('SYSTEM', 'SYSTEM_STATISTICS', 'TOOLS');
PL/SQL procedure successfully completed.

SQL> EXECUTE DBMS_STATS.GATHER_SYSTEM_STATS ( -
>     interval => 180, -
>     stattab => 'SYSTEM_STATISTICS', -
>     statid => '05-MAR-02' -
> );
PL/SQL procedure successfully completed.
```

The above commands would create a table (i.e., SYSTEM_STATISTICS) to hold the system statistics, populate that table with statistics gathered over a three-hour (180-minute) period, and assign an identifier of 05-MAR-02 to those statistics.

The init.ora parameter JOB_QUEUE_PROCESSES must be set to a non-zero value before using the INTERVAL option of the GATHER_SYSTEM_STATS procedure.

Alternatively, you can also manually determine the system statistics gathering interval by interactively entering the following commands:

```sql
SQL> EXECUTE DBMS_STATS.GATHER_SYSTEM_STATS ( -
>     gathering_mode => 'START', -
>     stattab => 'SYSTEM_STATISTICS' -
> );
PL/SQL procedure successfully completed.

SQL> EXECUTE DBMS_STATS.GATHER_SYSTEM_STATS ( -
>     gathering_mode => 'STOP' -
> );
PL/SQL procedure successfully completed.
```
If no table name is specified with the STATTAB parameter, then the gathered system statistics will be stored in the data dictionary. Users who wish to store statistics in the data dictionary must have SYSDBA privileges.

Oracle highly recommends that system statistics be gathered in order to improve the performance of the CBO.

For complete details of the many options available in the DBMS_STATS package, see Chapter 63, “DBMS_STATS,” of Oracle9i Supplied PL/SQL Packages and Types Reference, Release 1 (9.0.1) (Part Number A89852-02).

Using the OEM Console to Gather Statistics

The OEM Console can also be used to gather schema statistics. By selecting Tools > Database Wizards > Analyze from the OEM Console menu, the Analyze Wizard shown in Figure 3.8 will be displayed.
All analyze options, including COMPUTE, ESTIMATE, and VALIDATE, are available in the Analyze Wizard. By default, the Analyze Wizard uses the DBMS_STATS package to perform its statistical analysis.

**Viewing Table and Index Statistics**

Once statistics have been gathered using one of the previously defined methods, you can examine them by querying several data dictionary views. Except for tables that are using the dynamic monitoring feature of the DBMS_STATS package, these views will only contain the statistics as they existed at the time they were gathered.

Statistics for analyzed tables can be found in the DBA_TABLES data dictionary view. Figure 3.9 shows a query that demonstrates the difference between the statistics in DBA_TABLES before and after the DBMS_STATS.GATHER_SCHEMA_STATS command is used to analyze the tables in a schema called HR.

![Figure 3.9 Unanalyzed vs. Analyzed Tables](image)

```
SQL> SELECT table_name, num_rows, blocks, empty_blocks, avg_row_len
  2   FROM dba_tablespaces
  3   WHERE owner = 'HR';

<table>
<thead>
<tr>
<th>TABLE_NAME</th>
<th>NUM_ROWS</th>
<th>BLOCKS</th>
<th>EMPTY_BLOCKS</th>
<th>AVG_ROW_LEN</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEPARTMENTS</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EMPLOYEES</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>JOBS</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>JOB_HISTORY</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LOCATIONS</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>REGIONS</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

SQL> EXEC DBMS_STATS.GATHER_SCHEMA_STATS ('HR');

```
SQL> SELECT table_name, num_rows, blocks, empty_blocks, avg_row_len
  2   FROM dba_tablespaces
  3   WHERE owner = 'HR';

<table>
<thead>
<tr>
<th>TABLE_NAME</th>
<th>NUM_ROWS</th>
<th>BLOCKS</th>
<th>EMPTY_BLOCKS</th>
<th>AVG_ROW_LEN</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEPARTMENTS</td>
<td>27</td>
<td>1</td>
<td>0</td>
<td>20</td>
</tr>
<tr>
<td>EMPLOYEES</td>
<td>108</td>
<td>2</td>
<td>0</td>
<td>66</td>
</tr>
<tr>
<td>JOBS</td>
<td>19</td>
<td>1</td>
<td>0</td>
<td>33</td>
</tr>
<tr>
<td>JOB_HISTORY</td>
<td>10</td>
<td>1</td>
<td>0</td>
<td>53</td>
</tr>
<tr>
<td>LOCATIONS</td>
<td>23</td>
<td>1</td>
<td>0</td>
<td>48</td>
</tr>
<tr>
<td>REGIONS</td>
<td>4</td>
<td>1</td>
<td>0</td>
<td>14</td>
</tr>
</tbody>
</table>
```

A description of the contents of each of the columns that stores statistics in the DBA_TABLES data dictionary view is included in Table 3.11.

Index statistics are found in the DBA_INDEXES data dictionary view. Figure 3.10 shows a query that demonstrates the difference between the statistics in DBA_INDEXES before and after the ANALYZE INDEX command is used to analyze an index called EMP_DEPARTMENT_INDEX.
A description of the contents of each of the columns that stores statistics in the `DBA_INDEXES` data dictionary view is included in Table 3.12.
The value for CLUSTERING_FACTOR is used by the CBO to estimate how many logical I/Os would have to be performed in order to retrieve all the table data via the index.

Even greater granularity of the contents of tables and their indexed columns can be found in the DBA_TAB_COL_STATISTICS data dictionary view. A description of the contents of each of the columns that stores statistics in the DBA_TAB_COL_STATISTICS data dictionary view is included in Table 3.13.

Note that the DBA_TAB_COL_STATISTICS data dictionary view lacks an owner column to distinguish between tables from two schemas that have the same name. Use the USER_TAB_COL_STATISTICS data dictionary view to see each table’s column statistics.
Table and index statistics need to be kept current in order for the cost-based optimizer to perform effectively. Out-of-date statistics that no longer reflect the actual underlying table data may cause the CBO to select less than optimal execution plans.

A table will not have entries in DBA_TAB_COL_STATISTICS until it has been analyzed.
Once statistics have been gathered using the above techniques, they can be copied from one database to another in order to achieve consistency between production and development environments.

**Oracle Objective**

*Copy statistics between databases*

**Copying Statistics between Databases**

One of the common drawbacks of development environments is that they do not accurately reflect the volume of data that will be used in the final production systems. This can lead to unexpectedly poor performance when applications are moved from development to production. To minimize this problem, Oracle9i allows you to copy your optimizer statistics from your production system to a smaller development system. Even though the data volumes on the development system may be smaller, the optimizer behaves the same way it will when presented with the data volume of the production system. This is extremely useful for developers because they can now see the execution plans that will be used in the production environment once the application is migrated.

This statistics migration process is accomplished by exporting the statistics from the production system and then importing them into the development system. The following steps show you how to use the `DBMS_STATS` package to perform the export and import operations:

1. In the production database, create the STATS table in the JOE schema in the TOOLS tablespace. This table will be used to hold the production schema statistics.

   ```sql
   SQL> EXECUTE DBMS_STATS.CREATE_STAT_TABLE
       2  ('JOE','STATS','TOOLS');
   ```

2. Capture the current statistics for Joe’s schema and store them in the newly created STATS table.

   ```sql
   SQL> EXECUTE DBMS_STATS.EXPORT_SCHEMA_STATS
       2  ('JOE','STATS');
   ```

3. Use the Oracle Export utility from the OS command line to export the contents of the STATS table from the production database.

   ```shell
   $ exp joe/sidney@PROD file=stats.dmp tables=(STATS) log=stats.log
   ```
4. Move the export dump file from the production server to the Development server using FTP.

   $ ftp devl
   Connected to DEVL
   Name (DEVL:jjohnson): johnson
   331 Password required for johnson.
   Password:
   230 User johnson logged in.
   Remote system type is Unix.
   Using binary mode to transfer files.
   ftp> put stats.dmp
   200 PORT command successful.
   150 Opening BINARY mode data connection for stats.dmp
   226 Transfer complete.
   ftp> bye

5. In the Development database, create the STATS table in the JOE schema in the TOOLS tablespace. This table will hold the exported contents of the STATS table from the production database.

   SQL> EXECUTE DBMS_STATS.CREATE_STAT_TABLE
   2  ('JOE', 'STATS', 'TOOLS');

6. Use the Oracle Import utility from the operating system command line to import the STATS dump file created on the production server into the STATS table on the Development server.

   $ imp joe/sidney@DEVL file=stats.dmp log=stats.log full=y

7. Move the statistics in Joe’s STATS table into the Development database’s data dictionary.

   SQL> EXECUTE DBMS_STATS.IMPORT_SCHEMA_STATS
   2  ('JOE', 'STATS');

---

Setting the Optimizer Mode

Now that you know how the rule-based and cost-based optimizers differ, understand the importance of timely statistics for proper CBO operation, and how to move statistics between databases, you’re ready to configure your
database to run in the optimizer mode of your choosing. The optimizer mode can be set at the following three levels:

- Instance level
- Session level
- Statement level

The following sections discuss each of these methods of setting the optimizer mode.

NOTE
The CBO’s behavior varies slightly from one release of Oracle to another. Oracle9i includes a new init.ora parameter, OPTIMIZER_FEATURES_ENABLE, which can be used to specify the Oracle CBO version you wish to use. For example, setting OPTIMIZER_FEATURES_ENABLE=8.1.7 will make the Oracle9i instance use the Oracle8i CBO.

**Optimizer Mode: Instance Level**

The optimizer mode is set at the instance level through the use of theOPTIMIZER_MODE init.ora parameter. Once set, this parameter will determine the default optimizer mode used by all sessions operating against the instance. The OPTIMIZER_MODE parameter can be set to one of the following five values:

- RULE
- CHOOSE
- FIRST_ROWS
- FIRST_ROWS_n
- ALL_ROWS

**RULE Mode**

Setting OPTIMIZER_MODE=RULE in the init.ora places the instance into rule-based optimization mode. The presence or absence of statistics has no effect on the optimizer while operating in this mode unless SQL hints are used (hints are discussed later in this section). This optimizer mode is included for backward compatibility and is not recommended for new development.
**CHOOSE Mode**

Setting OPTIMIZER_MODE=CHOOSE in the init.ora places the instance into either rule-based optimization mode or cost-based optimization mode. If OPTIMIZER_MODE=CHOOSE and statistics exist for any one of the tables involved in a query, then the instance will use cost-based optimization when executing that query. Any missing statistics will be estimated “on the fly” by the optimizer. If OPTIMIZER_MODE=CHOOSE and no statistics exist on any tables or indexes in the query, then the instance will use rule-based optimization when executing the query. CHOOSE is the default setting for OPTIMIZER_MODE in the init.ora.

Some Oracle features require the use of the CBO. For example, even if the instance optimizer mode is set to RULE, the CBO will be used any time one of these features is used: Partitioned tables and indexes, index-organized tables, reverse key indexes, function-based indexes, bitmap indexes, and query rewrite and materialized views. Several of these features are discussed later in this chapter.

**FIRST_ROWS Mode**

Setting OPTIMIZER_MODE=FIRST_ROWS in the init.ora also places the instance into either rule-based or cost-based optimization mode based on the presence or absence of table and index statistics. In this mode, if statistics exist, then the instance will use a variation of CBO designed to return the first rows fetched by a query as quickly as possible. This mode is designed to improve the overall response time of queries.

**FIRST_ROWS_n Mode**

Like FIRST_ROWS, FIRST_ROWS_n is also designed to improve overall response times of queries. However, by specifying a value of 1, 10, 100, or 1,000 for n, you can specifically optimize the response time for the first n number of rows returned by a query.

**ALL_ROWS Mode**

Setting OPTIMIZER_MODE=ALL_ROWS in the init.ora also places the instance into either rule-based or cost-based optimization mode based on the presence or absence of table and index statistics. In this mode, if statistics exist, then the instance will use a variation of CBO designed to return all the rows fetched by a query as quickly as possible. This mode is used to improve the overall throughput of queries. ALL_ROWS is the default CBO
mode. ALL_ROWS is the default optimizer mode used when OPTIMIZER_MODE is set to CHOOSE in the init.ora.

**Optimizer Mode: Session Level**

Once the init.ora optimizer mode is set at the instance level, all users connecting to the instance will utilize that optimization mode as the default for processing their queries. However, it is possible for a user to change the optimizer mode for their session using only the ALTER SESSION command. The following example shows the current session being changed to cost-based optimization mode:

```sql
SQL> ALTER SESSION SET OPTIMIZER_MODE=CHOOSE;
```

The possible values for the optimizer mode at the session level are the same as those at the instance level: RULE, CHOOSE, FIRST_ROWS, FIRST_ROWS_n, and ALL_ROWS. This optimizer mode is retained by the session until the user either resets it using the ALTER SESSION command again or logs out.

**Optimizer Mode: Statement Level**

The optimizer can also be influenced at the statement level through the use of hints. Hints are specific instructions that make the optimizer perform in a particular way.

<table>
<thead>
<tr>
<th>Oracle Objective</th>
<th>Describe how hints are used</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Hints are embedded within the actual SQL statement itself by surrounding the hint with the symbols /*+ and */. An example of a hint that would make the specified SQL statement run in rule-based optimization mode is shown here:</td>
</tr>
</tbody>
</table>

```sql
SQL> SELECT /*+ RULE */ dist.distributor_id,  
2  dist.city,  
3  dist.state,  
4  dist.zip_code,  
5  district.name,  
6  emp.last_name  
7 FROM distributor dist, district, employee emp  
8 WHERE emp.employee_id = dist.manager_id  
9 AND district.district_id = dist.district_id;
```
Make sure you have the + immediately after the /* when specifying a hint. If you have a space or any other characters than /*+ preceding the hint, Oracle will treat the hint specification as a comment and ignore it.

Four of the instance- and session-level optimizer modes you have already seen (RULE, FIRST_ROWS, FIRST_ROWS_n, and ALL_ROWS) can also be specified as hints. These are the only hints that influence the optimizer mode. In addition to these, there are also nearly 40 other hints available in Oracle9i. It is beyond the scope of this text and the Oracle9i Performance Tuning OCP exam to discuss them all in detail. However, some more commonly used hints are described in Table 3.14.

### Table 3.14 Commonly Used Optimizer Hints

<table>
<thead>
<tr>
<th>Hint Name</th>
<th>Example</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FULL</td>
<td>/*+ FULL SALES */</td>
<td>Forces optimizer to use a full table scan when accessing the SALES table</td>
</tr>
<tr>
<td>INDEX</td>
<td>/*+ INDEX SALES_ID_PK */</td>
<td>Forces the optimizer to use the index SALES_ID_PK when accessing the SALES table the index is built on</td>
</tr>
<tr>
<td>REWRITE</td>
<td>/*+ REWRITE */</td>
<td>Forces the optimizer to dynamically rewrite the query in order to make use of a materialized view if one is available</td>
</tr>
<tr>
<td>PARALLEL</td>
<td>/*+ PARALLEL */</td>
<td>Causes the specified operation to execute in parallel</td>
</tr>
</tbody>
</table>

All hints, with the exception of RULE, cause the CBO to be used when the query is executed.

A complete list of all the Oracle hints can be found in Chapter 5, “Optimizer Hints,” in the Oracle9i Database Performance Guide and Reference Release 1 (9.0.1) (Part Number A87503-02).
Optimizer Precedence Rules

Since the optimizer mode can be set at the instance, session, or statement level in the form of a hint, it is important to understand which setting will take precedence when a statement is actually executed:

- Statement level hints override session and instance level optimizer modes.
- Session level optimizer settings override instance level optimizer settings.
- Instance level settings take effect if no session level optimizer settings or hints are specified.
- The default instance optimizer mode is CHOOSE. If statistics exist on any table or index involved in a query, then CBO is used. If no statistics exist, then RBO is used.

Improving Application Performance

Gathering and analyzing application information in the form of trace files and execution plans is the first step to understanding how your application is performing. Knowing how the application is affected by the behavior of the Oracle optimizer is important as well. Drawing on this information, the next sections will examine the options available to you for improving the performance of an application. These techniques fall into two broad categories: improving execution paths by storing optimal plans and using materialized views, and minimizing I/O by using indexes and clusters.

Improving Execution Paths

Depending on the complexity of the query, the size of the tables, and the number of alternatives considered by the Oracle optimizer, a less-than-optimal execution plan may be chosen by the CBO, leading to degraded query performance. Oracle9i uses two features—plan stability and materialized views—to help you improve performance by directly influencing the CBO’s choice of an execution plan.

| Oracle Objective | Explain the concept of plan stability |
Plan Stability

Changes that you make to the database may inadvertently affect execution plans. For example, changes made to init.ora parameters to improve one aspect of database tuning may cause the Oracle optimizer to behave differently than it did before—possibly to the detriment of previously well-tuned SQL statements. Oracle9i allows you to address this problem by giving specified execution plans “stability.” Plan stability is accomplished by storing the preferred execution plan for a given SQL statement in the data dictionary, and then utilizing that execution plan whenever that SQL statement is issued.

Stored Outlines

Oracle9i maintains these predefined execution plans in the data dictionary in the form of stored outlines. These stored outlines represent the execution plan the optimizer will use whenever the SQL statement associated with the stored outline is executed. This stored execution plan remains constant even if new statistics are gathered or optimizer-related changes are made to the init.ora. A stored outline is essentially a collection of hints that will reproduce the execution plan that the optimizer used when it originally executed a statement.

Through a process known as plan equivalence, a SQL statement must exactly match the original statement used to generate an outline in order for the stored outline to be used. Even an identical statement with the addition of a hint or comment will not be considered equivalent for stored outline usage.

Creating Stored Outlines

Several configuration steps must be performed before using the shared outlines feature. These steps are listed here in the order in which they should be performed:

1. Determine which statements you would like to store execution plans for. It is likely that these statements will involve tables and indexes whose data is relatively static. Queries that utilize tables and indexes of highly volatile data will probably benefit from the normal, dynamic nature of the CBO. If you wish to generate stored outlines for every
Improving Application Performance

statement, simply set the init.ora parameter CREATE_STORED_OUTLINES=TRUE. This option should be used sparingly since it is unlikely that stored outlines are appropriate for every SQL statement. If this option is used, steps 2–4 do not apply.

2. Ascertain whether the statements you have identified can be lumped into two or more broad categories of statements. Doing so will help you determine whether a statement is eligible for stored outline execution. Examples of categories might include “Queries that access the EMPLOYEE table” or “Queries that do a lookup on the ADDRESS column of the DISTRIBUTOR table.” At outline creation, if you do not specify which category a stored outline should belong to, it will be stored in a category called DEFAULT. The stored outline category can be specified as follows:

1. At the instance level using the USE_STORED_OUTLINES parameter in the init.ora. Setting USE_STORED_OUTLINES=TRUE automatically causes Oracle to store outlines in the DEFAULT category.

2. At the session level using the ALTER SESSION SET USE_STORED_OUTLINE command.

3. Alter the OUTLN schema. This schema stores all the stored outlines for the database.

   - For security purposes, change the OUTLN schema’s password. The default password (OUTLN) is well documented. Oracle9i automatically prompts you to change the OUTLN user’s password after the first login.

   - Alter the OUTLN schema’s default and temporary tablespaces to something other than the SYSTEM tablespace.

   - Move the two tables that the OUTLN schema uses to store outlines (OL$ and OL$HINTS) out of the SYSTEM tablespace and into OUTLN’s new default tablespace. The Export and Import utility, or the ALTER TABLE ... MOVE... command can be used for this operation.

4. To create the stored outlines for the statements you identified, use the following commands:

   SQL> CREATE OR REPLACE OUTLINE employee_mgr_last_name
   2   FOR CATEGORY employee_queries ON
   3   SELECT lastname
   4   FROM employee
   5   WHERE job_title = 'MGR';
where EMPLOYEE_MGR_LAST_NAME is the name of the new stored outline and EMPLOYEE_QUERIES is the name of the category to which it is assigned. Alternatively, you can activate outlines for a particular category in your session and then issue each of the commands you wish to create outlines for:

```
SQL> ALTER SESSION SET CREATE_STORED_OUTLINES = employee_queries;
SQL> SELECT lastname
2  FROM employee
3  WHERE job_title = 'MGR';
SQL> ALTER SESSION SET CREATE_STORED_OUTLINES = false;
```

where EMPLOYEE_QUERIES is the name of the category that this stored outline will be assigned. The name of the stored outline will be system generated.

You can also use the Outline Manager GUI tool from the Enterprise Manager Tuning Pack to create your stored outline. Figure 3.11 shows the Outline Manager screen being used to create an outline called EMPLOYEE_STATE.

**FIGURE 3.11** Enterprise Manager's Outline Manager
5. Next, activate the stored outlines feature using one of the following three methods:
   - Set the `init.ora` parameter `USE_STORED_OUTLINES=TRUE` and restart the instance.
   - Place the instance into stored outlines mode dynamically using:
     
     ```
     SQL> ALTER SYSTEM SET USE_STORED_OUTLINES = true;
     ```

   - Turn on the stored outline feature at the session level by using:
     
     ```
     SQL> ALTER SESSION SET USE_STORED_OUTLINES = true;
     ```

   The above command will activate stored outlines in all categories. To activate categories selectively use: `ALTER SYSTEM SET USE_STORED_OUTLINES = employee_queries;`

   - If a stored outline for an SQL statement exists in the `DEFAULT` category, but no corresponding outline exists in the active category, the `DEFAULT` stored outline will still be used.

**How Stored Outlines Are Used**

When a SQL statement is executed, Oracle first compares the statement and the session-level or instance-level outline category to the statements and categories already cached in the Shared Pool. If a match is found in the Shared Pool the statement is executed using the cached information. If a matching entry is not found in the Shared Pool, Oracle next examines the data dictionary to see if a stored outline exists for this statement and this category. If a stored outline does exist in the data dictionary, the optimizer uses the stored execution plan to perform the query. If no stored outline exists in the data dictionary, the CBO will dynamically generate an execution plan as it normally would.
Managing Stored Outlines

Stored outline requirements change over time. Outlines may need to be modified, dropped, renamed, or moved from one category to another. These types of maintenance operations are performed using the Outline Manager GUI tool, the Oracle-supplied package `OUTLN_PKG`, and the `ALTER OUTLINE` and `DROP OUTLINE` commands.

**USING OUTLINE MANAGER**

In addition to creating stored outlines, the Outline Manager GUI tool can also be used to modify existing stored outlines. Figure 3.12 shows the Outline Editor being used to change the access method (via the `FULL` hint) of the stored outline called `EMPLOYEE_STATE`.

The Outline Editor can also be used to modify the join order of the SQL statement behind the stored outline, change the category that the stored outline is assigned to, or drop a stored outline.

**FIGURE 3.12** Editing a stored outline using Outline Manager

**USING OUTLN_PKG**

The `OUTLN_PKG` package contains three procedures that can be used to manage stored outlines. The names and descriptions of these procedures are shown in Table 3.15.
The following examples demonstrate the use of each of these procedures:

```sql
SQL> EXECUTE OUTLN_PKG.DROP_UNUSED;
SQL> EXECUTE OUTLN_PKG.DROP_UNUSED ('EMPLOYEE_QUERIES');
SQL> EXECUTE OUTLN_PKG.UPDATE_BY_CAT ('DEFAULT','DSS_QUERIES');
```

**USING ALTER AND DROP OUTLINE**

The ALTER OUTLINE and DROP OUTLINE commands can also be used to manage stored outlines. Table 3.16 shows the available options for the ALTER OUTLINE command.

**TABLE 3.15** Procedures in OUTLN_PKG

<table>
<thead>
<tr>
<th>Procedure Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DROP_UNUSED</td>
<td>Drops outlines in current schema that have not been used since they were created</td>
</tr>
<tr>
<td>DROP_BY_CAT</td>
<td>Drops the specified category and all outlines contained in it</td>
</tr>
<tr>
<td>UPDATE_BY_CAT</td>
<td>Creates the new category name specified and moves all the outlines in the old category specified into the new category</td>
</tr>
</tbody>
</table>

**TABLE 3.16** Options for ALTER OUTLINE

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>REBUILD</td>
<td>Generates a new execution plan for the specified stored outline using current statistics</td>
</tr>
<tr>
<td>RENAME TO</td>
<td>Changes the name of the stored outline to the new name specified</td>
</tr>
<tr>
<td>CHANGE CATEGORY TO</td>
<td>Changes the category to which the stored outline belongs</td>
</tr>
</tbody>
</table>

The following examples demonstrate the use of each of these options:

```sql
SQL> ALTER OUTLINE employee_mgr_last_name REBUILD;
SQL> ALTER OUTLINE employee_mgr_last_name RENAME TO employee_lname;
SQL> ALTER OUTLINE employee_lname CHANGE CATEGORY TO emp_queries;
```
If an outline is no longer needed, you can use the DROP OUTLINE command to delete it from the system. An example of this syntax is shown here:

```
SQL> DROP OUTLINE employee_lname;
```

If the stored outline that is dropped is the last outline in its category, the category will also be dropped. Oracle does not maintain categories that do not contain stored outlines.

**Viewing Stored Outline Data**

The data dictionary views DBA_OUTLINE_HINTS and DBA_OUTLINES contain information about all the stored outlines in the database. A description of the contents of each of these views is contained in Table 3.17 and Table 3.18.

**TABLE 3.17** Contents of the *DBA_OUTLINES* View

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAME</td>
<td>The name of the stored outline</td>
</tr>
<tr>
<td>OWNER</td>
<td>The schema name of the stored outline’s owner</td>
</tr>
<tr>
<td>CATEGORY</td>
<td>The category the stored outline is associated with</td>
</tr>
<tr>
<td>USED</td>
<td>Indicates whether the outline has ever been used (values are USED/UNUSED/UNDEFINED)</td>
</tr>
<tr>
<td>TIMESTAMP</td>
<td>Date and time the stored outline was created</td>
</tr>
<tr>
<td>VERSION</td>
<td>Oracle RDBMS version that created the stored outline</td>
</tr>
<tr>
<td>SQL_TEXT</td>
<td>The actual text of the original SQL query associated with the stored outline</td>
</tr>
</tbody>
</table>

**TABLE 3.18** Contents of the *DBA_OUTLINE_HINTS* View

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAME</td>
<td>The name of the stored outline</td>
</tr>
<tr>
<td>OWNER</td>
<td>The schema name of the stored outline’s owner</td>
</tr>
</tbody>
</table>
Private Outlines

Using the above techniques to alter a stored outline will cause the outline’s behavior to change for all users of the instance. However, users may occasionally want to make changes to stored outlines that will impact only their session. This can be accomplished using private outlines. A **Private Outline** is an outline that resides within an individual user’s session. As such, any changes made to the outline will affect only that session while all other sessions will continue to use the unaltered public stored outline.

Before using private outlines, tables must be created in the user’s schema to hold the private copy of the public stored outline. These tables can be built by executing the `DBMS_OUTLN_EDIT.CREATE_EDIT_TABLES` procedure. Additionally, the user who wishes to create private outlines must also have the `CREATE ANY OUTLINE` privilege as well as `SELECT` privileges on the tables involved in the outline’s query. Once these prerequisites are met, you can use the following steps to create a private outline. This example creates a private outline called `PRIVATE_OL` from a public stored outline called `PUBLIC_OL`:

1. Copy the public stored outline’s definition into the private outline using the `CREATE PRIVATE OUTLINE private_ol FROM public_ol` command.

2. Alter the private outline using one of the techniques discussed in the previous section and then re-sync the outline using the `CREATE PRIVATE OUTLINE private_ol FROM PRIVATE private_ol` command.
3. Activate private outlines in the session using the ALTER SESSION SET USE_PRIVATE_OUTLINES=TRUE command.

4. Execute the query referenced in the outline and then use the EXPLAIN PLAN or TKPROF utilities to examine the effectiveness of the new private outline.

5. If you decide that you would like to make your private outline available to all other users, you can make it public by issuing the CREATE OR REPLACE OUTLINE public_ol FROM PRIVATE private_ol command.

6. Deactivate private outlines in the session using the ALTER SESSION USE_PRIVATE_OUTLINES(FALSE command.

Materialized Views

Materialized Views

Stored outlines help speed up queries by telling the optimizer how to tackle the query execution associated with a particular SQL statement. Materialized views are also designed to speed up queries by storing data from queries in a pre-joined, pre-summarized format. Unlike a traditional view, which is merely stored in the data dictionary as a SELECT statement that is executed when the view is accessed, a materialized view stores the physical results of the view in its own segment, separate and distinct from the underlying table on which the view is based. This segment can be stored in its own tablespace and can be indexed and partitioned.

Oracle Objectives

- Explain different storage structures and why one storage structure might be preferred over another
- Describe Materialized Views and the use of Query Rewrites

Materialized views are intended primarily for use in data warehouses and Decision Support Systems where large volumes of data are accessed and summarized using queries. As with a stored outline, the optimizer recognizes when the use of a materialized view is appropriate and dynamically adjusts the query to make use of the associated view.
However, unlike a stored outline, a query does not have to match exactly the SQL statement used to create the materialized view. Instead, the optimizer can dynamically rewrite a query that is close to the original definition so that the materialized view can be used in place of the actual tables. This query rewrite functionality happens automatically and is transparent to the user issuing the query.

You can use the EXPLAIN_MV and EXPLAIN_REWRITE packages in the DBMS_MVIEW package in order to identify why Oracle did, or did not, rewrite a particular query to make use of an available materialized view. See Chapter 29, “DBMS_MVIEW,” in the Oracle9i Supplied PL/SQL Packages and Types Reference Release 1 (9.0.1) (Part Number A89852-02) for more information.

Using Materialized Views
Several configuration steps must be performed before using the materialized view feature. These steps are listed here in the order in which they should be performed:

1. Determine the statements for which you would like to create materialized views. It is likely that these statements will involve large fact tables that are joined to one another and include summary functions like COUNT, MAX, MIN, or SUM.

Oracle provides a Summary Advisor utility that can be used to identify SQL statements that may make good candidates for materialized views. Summary Advisor also helps identify existing materialized views that are underutilized and could possibly be dropped. See Chapter 16, “Summary Advisor,” in the Oracle9i Data Warehousing Guide Release 1 (9.0.1) (Part Number A90237-01) for more information.

2. Decide whether or not you wish to keep the data in the view in sync with the underlying base tables. If you do not want to keep the data in sync, you will specify the NEVER option during the creation of the view. If you do want the data to be kept in sync, you will specify either the
Chapter 3 · SQL Application Tuning and Design

COMPLETE, FAST, or FORCE option during the creation of the view. These options will affect how the data contained in the materialized view is refreshed:

- **COMPLETE**: During a refresh, the materialized view is truncated and then completely repopulated with data from the base tables in the query.
- **FAST**: During a refresh, the materialized view is populated only with data that has changed in the base table since the last re-sync. This refresh is performed using the view’s log data or by ROWID.
- **FORCE**: Oracle attempts to refresh the view using the Fast refresh method first. If the Fast refresh method is not available, then the Complete refresh is used. Force is the default sync mode.

3. Next, determine the frequency with which you would like your chosen refresh process to occur. There are three possible refresh modes:

- **ON COMMIT**: The materialized view data will be refreshed with every committed transaction on the base tables.
- **By Time**: By using the START WITH and NEXT options during view creation, the materialized view data can be refreshed at specified times on specified dates. These refresh times and dates are submitted as jobs to the Oracle job queue.
- **ON DEMAND**: You can also manually refresh the contents of a materialized view.

4. Set the `init.ora` parameters related to materialized views. A list of these parameters and their descriptions is in Table 3.19.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>JOB_QUEUE_PROCESSES</td>
<td>The number of background job queue processes to start—must be set to a value of 1 or more when using materialized views.</td>
</tr>
<tr>
<td>QUERY_REWRITE_ENABLED</td>
<td>Allows optimizer to dynamically rewrite queries to take advantage of materialized views when set to TRUE.</td>
</tr>
</tbody>
</table>
The `QUERY_REWRITE_ENABLED` and `QUERY_REWRITE_INTEGRITY` parameters can also be set at the session level using the `ALTER SESSION` command. The system privilege `GLOBAL QUERY REWRITE` and/or `QUERY REWRITE` must be granted to users who wish to alter their session in this manner. There are also two SQL hints, `REWRITE` and `NOREWRITE`, which can also be used to turn on and off the dynamic rewriting of queries when using materialized views.

Users need privileges only on the base tables underlying the materialized view in order to use it, not on the view itself.

**Creating Materialized Views**

Once you have identified the SQL statements for which you would like to create materialized views, you are ready to build the materialized views. The sample code below shows how a materialized view would be built on the `DISTRICT`, `DISTRIBUTOR`, and `EMPLOYEE` tables:

```
SQL> CREATE MATERIALIZED VIEW EMP_BY_DISTRICT
  2  TABLESPACE MVIEW_DATA
  3  BUILD IMMEDIATE
  4  REFRESH FAST
```
The above command will create a new materialized view called EMP_BY_DISTRICT. At a refresh, the materialized view will be refreshed with only the data that has been added since the last refresh. The materialized view will also automatically make use of Oracle’s query rewrite functionality in order to improve the usability of the view. The materialized view will be built with the BUILD IMMEDIATE option. This is one of three options that can be specified during the creation of the materialized view:

- BUILD IMMEDIATE causes Oracle to build the materialized view and populate it with data immediately after the command is executed.
- BUILD DEFERRED causes Oracle to build the structure of the materialized, but does not populate the materialized view with data until the first refresh occurs.
- ON PREBUILT TABLE causes Oracle to use a preexisting Oracle table that already contains the data that is in the view definition, rather than building a new structure to hold the same data.

Managing Materialized Views

Once your materialized views are created, you will need to know how to refresh, disable, and drop them.

REFRESHING MATERIALIZED VIEWS

Materialized views can be set up to refresh manually or automatically. Manual refreshes are performed with the DBMS_MVIEW package. Automatic refreshes are accomplished by creating the materialized view with the COMMIT option or by scheduling the refresh using the DBMS_MVIEW package.

Refreshes can be generated manually using the DBMS_MVIEW package and one of its three procedures: REFRESH, REFRESH_DEPENDENT, REFRESH_ALL_MVIEWS. Each of these procedures is described in Table 3.20.
The following examples demonstrate the use of each of these procedures:

```sql
SQL> EXECUTE DBMS_MVIEW.REFRESH ('EMP_BY_DISTRICT');

SQL> EXECUTE DBMS_MVIEW.REFRESH_DEPENDENT ('EMPLOYEE');

SQL> EXECUTE DBMS_MVIEW.REFRESH_ALL_MVIEWS;
```

A materialized view that was created with the REFRESH FAST ON COMMIT or REFRESH COMPLETE ON COMMIT option will be refreshed automatically after each commit. You could also create the views with the REFRESH FAST ON DEMAND or REFRESH COMPLETE ON DEMAND option and then demand their refresh using the Oracle-supplied PL/SQL packages DBMS_MVIEW and DBMS_JOB. The submission of a refresh job for a materialized view called EMP_BY_DISTRICT that executes every Sunday at 2:00 A.M. can be done using the PL/SQL procedure shown below:

```sql
CREATE OR REPLACE PROCEDURE refresh_emp_by_district
IS
    v_job_num NUMBER;
BEGIN
    DBMS_JOB.SUBMIT(
        v_job_num,
        'DBMS_MVIEW.REFRESH (''EMP_BY_DISTRICT'''');
        SYSDATE,
        'NEXT_DAY(TRUNC(SYSDATE),''SUNDAY'') + 2/24'
    );
END;
/
```

**TABLE 3.20** Procedures for Manual Refresh of Materialized Views

<table>
<thead>
<tr>
<th>Procedure Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>REFRESH</td>
<td>Refreshes the specified materialized view</td>
</tr>
<tr>
<td>REFRESH_DEPENDENT</td>
<td>Refreshes all materialized views that utilize this table</td>
</tr>
<tr>
<td>REFRESH_ALL_MVIEWS</td>
<td>Refreshes all materialized views in the schema that have not been refreshed since the last bulk load of data to the base tables</td>
</tr>
</tbody>
</table>
DISABLING MATERIALIZED VIEWS
Materialized views can be disabled at the instance, session, or statement level by using one of the following four methods:

- Set the init.ora parameter QUERY_REWRITE_ENABLED=FALSE and restart the instance.
- Dynamically set the value of QUERY_REWRITE_ENABLED=FALSE using the ALTER SYSTEM command.
- Set QUERY_REWRITE_ENABLED=FALSE at the session level using the ALTER SESSION command.
- At the statement level, use the NOREWRITE hint in a query.

Users need the QUERY REWRITE or GLOBAL QUERY REWRITE system privilege in order to enable and disable the query rewrite functionality on materialized views in their own schema, or another user’s schema, respectively.

DROPPING MATERIALIZED VIEWS
The following example demonstrates the syntax for dropping a materialized view:

```sql
SQL> DROP MATERIALIZED VIEW emp_by_district;
Materialized view dropped.
```

Dropping a materialized view does not affect any of the data in the base tables that the view was built on.

Minimizing I/O Using Indexes and Clusters
Stored outlines and materialized views aid query performance by reducing the options that the optimizer must consider when developing an execution plan. This section deals with the other half of the query execution equation—the speed with which the data is actually retrieved once the plan is executed. One of the most effective methods for reducing the time it takes for the Oracle Server to find and return rows is achieved through the use of indexes, partitions, and clusters.
Indexes

The appropriate use of indexing is probably the single most effective tuning tool in the DBA's arsenal. This is true because indexes cut down substantially on unnecessary I/O against large tables, which contributes greatly to the response time of a query. The six index types available in Oracle9i are:

- B-Tree index
- Compressed B-Tree
- Bitmap index
- Function-based Index
- Reverse Key Index (RKI)
- Index Organized Table (IOT)

B-Tree Indexes

Balanced tree, or B-Tree, indexes are the most common type of index found in modern relational databases. B-Tree indexes sort the column data of the table columns they are built on in ascending order. In addition to the sorted column data, the B-Tree index also stores a row ID indicating where the rest of that row’s data is stored in the indexed table. The index then stores these values in a tree-structured format.

For example, suppose your application had a table called EMPLOYEE that contained the following data:

```sql
SQL> SELECT *
2  FROM employee;
```

```
EMP_ID LAST_NAME FIRST_NAME TITLE DEPT_ID G
---------- ---------- ---------- ------------------- -------
100  Baker Brenda  Dir Web Development 601   F
101  Douglass Laura  Dir of Research 301   F
102  Schwarz Jim  Dir of IS 401   M
110  Block Robin  Corporate Counsel 501   F
106  Bron Reginald  Dir of Marketing 601   M
108  Weishan Matt  Technical Writer 601   M
107  Gastel Tony  Oracle DBA 401   M
```
By examining trace files and Explain Plans, you have determined that a B-Tree index on the LAST_NAME column of the table will likely help performance of querying against the EMPLOYEE table:

```sql
SQL> CREATE INDEX employee_last_name_idx
2  ON employee (last_name)
3  TABLESPACE APPL_IDX;
```

The resulting index would store the values for the LAST_NAME column, along with the row IDs for those values, in a B-Tree structure as shown in Figure 3.13.

**Figure 3.13** Index contents and structure for B-Tree index on LAST_NAME column of EMPLOYEE

By first searching the index, which is arranged in the multileveled tree structure, and then accessing the required table data by row ID, costly full-table scan operation is avoided and query performance is enhanced. B-Tree indexes are very useful when executing queries that will return only a small number of the overall rows found in a table. As a rule of thumb, Oracle recommends considering a B-Tree indexing strategy whenever queries frequently return less than 5 percent of the total rows in the table. The best candidates for B-Tree indexes are those columns having high cardinality in their data. These columns should also be frequently used in SQL statement WHERE clauses.

By design, B-Tree indexes are always balanced. This is true because the Oracle Server continually splits full index blocks as new values are added to the table that the index is built on. However, as insert and delete activity occurs on a table, its associated indexes can develop many levels. This tends to degrade the efficiency of the B-Tree index because it takes the Oracle Server longer to work its way from the root, or starting block, down to the
level that actually contains the index entry for the data in question. You can determine how "deep" an index is by analyzing the index and then performing the following query on DBA_INDEXES:

```sql
SELECT index_name, blevel
FROM dba_indexes
WHERE blevel >= 4;
```

The above query will display all the indexes that have four or more levels between their root block and their deepest leaf blocks. These indexes are good candidates for rebuild. Rebuilding these indexes will reduce the number of levels and improve the performance of the index. This same information can also be found in the OEM Console under the Databases > Schema > Indexes option. By right-clicking an index and selecting the View/Edit Details menu choice, the Statistics tab can be selected and the screen shown in Figure 3.14 will be displayed.

**FIGURE 3.14** OEM Schema option showing index statistics
The output in Figure 3.14 indicates that the JHIST_EMPLOYEE_INDEX index Tree Height is 2. This is less than the recommended height of 4 or more levels that should exist before an index should be considered a candidate for rebuilding.

Another indication that an index may benefit from a rebuild is evident when the deleted entries in the index represent more than 20 percent of the overall entries. When this is the case, the index is using more space than it needs to store index entries and is therefore increasing the number of blocks that must be accessed to retrieve an entry. The following SQL commands show how to identify indexes with this problem:

```sql
SQL> ANALYZE INDEX employee_last_name_idx VALIDATE STRUCTURE;
```

The VALIDATE STRUCTURE option for the ANALYZE command populates the data dictionary view INDEX_STATS with values. You can then query this view to see if any space is being wasted in the index:

```sql
SQL> SELECT (DEL_LF_ROWS_LEN/LF_ROWS_LEN) * 100
2  "Wasted Space"
3  FROM index_stats
4  WHERE name = 'EMPLOYEE_LAST_NAME_IDX';
```

<table>
<thead>
<tr>
<th>Wasted Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>34</td>
</tr>
</tbody>
</table>

The results of the above query indicate that 34 percent of the space in the index’s blocks is wasted because they used to contain entries for rows that have been deleted.

Using this calculation, Oracle recommends rebuilding any index whose wasted space exceeds 20 percent.

When either of the above queries indicates that an index’s performance will benefit from rebuilding, there are three rebuilding techniques that can be considered:

1. Drop and recreate the index from scratch.
2. Rebuild the index using the ALTER INDEX ... REBUILD command.
3. Rebuild the index using the ALTER INDEX ... COALESCE command.
While each of the above techniques will result in a more streamlined rebuilt index, each has its own advantages and disadvantages.

**Drop and Re-Create Index**  Dropping and re-creating an index is the most time-consuming and resource-intensive technique for rebuilding an index. However, in Oracle versions prior to 7.3 this was the only technique for rebuilding an index.

**Using the ALTER INDEX ... REBUILD Command**  Using the ALTER INDEX command with the REBUILD option is an effective way to quickly rebuild an index because the existing index entries are used to create the new index. The following is an example of the ALTER INDEX command being used to rebuild an index called EMP_LAST_NAME_IDX:

```sql
SQL> ALTER INDEX emp_last_namd_idx REBUILD ONLINE;
```

The ONLINE option of the ALTER INDEX...REBUILD command should be used to minimize any locking issues that can occur when an index is rebuilt while users continue to perform DML on the index’s underlying table.

Indexes can also be rebuilt using the COMPUTE STATISTICS, PARALLEL, UNRECOVERABLE, and NOLOGGING options.

Since both the old and the new index exist briefly during the rebuild process, using this technique to rebuild indexes requires that additional disk space be temporarily available. The ALTER INDEX ... REBUILD command can also be used to move an index to a new tablespace.

**Using the ALTER INDEX ... COALESCE Command**  Using the ALTER INDEX command with the COALESCE option is another effective way to quickly rebuild an index. The following is an example of the ALTER INDEX command being used to coalesce an index called EMP_BIRTH_DATE_IDX:

```sql
SQL> ALTER INDEX emp_birth_date_idx COALESCE;
```

Unlike the REBUILD option, the COALESCE option does not require any additional disk space during the rebuild operation. Instead, the COALESCE option coalesces the leaf blocks within the same index branch when rebuilding the index. This minimizes potential locking issues associated with the rebuild process. However, the COALESCE option cannot be used to move an index to a new tablespace.
For more information on the ALTER INDEX commands, see Chapter 16, “Managing Indexes,” in the Oracle9i Database Administrator’s Guide Release 1 (9.0.1) (Part Number A90117-01).

Compressed B-Tree Indexes
When B-Tree indexes are built on large tables, particularly in a data warehouse or a Decision Support System, the indexes can also consume large amounts of storage space. Compressed B-Tree indexes are used to minimize the amount of space used by certain types of B-Tree indexes. When a B-Tree index is compressed, duplicate occurrences of the indexed column value are eliminated—thus reducing the overall amount of space needed to store the index. Figure 3.15 shows a comparison of an uncompressed and compressed B-Tree index on an EMPLOYEE table where several of the employees had the last name of Smith.

![Uncompressed vs. compressed B-Tree index](image)

**FIGURE 3.15** Uncompressed vs. compressed B-Tree index

<table>
<thead>
<tr>
<th>Uncompressed B-Tree Index</th>
<th>Compressed B-Tree Index</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Last Name</strong></td>
<td><strong>Associated ROWID</strong></td>
</tr>
<tr>
<td>Smith</td>
<td>AAABSOAAEAAAAABTAAB</td>
</tr>
<tr>
<td>Smith</td>
<td>AAABSOAAEAAAAABTAAC</td>
</tr>
<tr>
<td>Smith</td>
<td>AAABSOAAEAAAAABTAAD</td>
</tr>
<tr>
<td>Smith</td>
<td>AAABSOAAEAAAAABTAEE</td>
</tr>
</tbody>
</table>

In the uncompressed index, each occurrence of the last name Smith also stores its associated ROWID entry. In the compressed index, the Smith entry is stored only once, with the ROWID entry for each of those associated rows appended to the name. Compressed indexes are built by adding the keyword COMPRESS to the end of the CREATE INDEX statement:

```
SQL> CREATE INDEX employee_hist_last_name_idx
    2  ON employee_history (last_name)
    3  TABLESPACE APPL_IDX
    4  COMPRESS;
```
Alternatively, non-compressed indexes can be compressed using the ALTER INDEX statement:

```
SQL> ALTER INDEX employee_hist_last_name_idx REBUILD COMPRESS;
```

It is not possible to build a compressed B-Tree index on a single table column containing unique values because no two index values would be the same.

**Bitmap Indexes**

B-Tree indexes work best on columns whose data has high cardinality. For columns with low cardinality, a Bitmap index may be a more effective indexing option. Unlike B-Tree indexes, Bitmap indexes create a binary mapping of the rows in the table and store that map in the index blocks. This means the resulting index will require significantly less space to store the index data and retrieve the rows of an equality match on the indexed column more quickly than an equivalent B-Tree index. This type of index is particularly useful on columns of large, low-DML activity tables that contain very few distinct values.

For example, if you decide from your examination of trace files and Explain Plans that an index should be built on the GENDER column of the EMPLOYEE table, a Bitmap index would be a good choice because the table is fairly large and the column’s data has low cardinality. This is true because only two possible values, M or F, are allowed for this column:

```
SQL> SELECT * 2 FROM employee;
```

```
EMP_ID LAST_NAME  FIRST_NAME TITLE              DEPT_ID GENDER
------ ---------- ----------  ------------------- ------------
100 Baker         Brenda     Dir Web Development    601 F
101 Douglass      Laura      Dir of Research        301 F
102 Jim           Schwarz    Dir of IS              401 M
110 Block         Robin      Corporate Counsel      501 F
106 Bron          Reginald   Dir of Marketing       601 M
108 Weishan       Matt       Technical Writer       601 M
107 Gastel        Tony       Oracle DBA             401 M
```

```
SQL> CREATE BITMAP INDEX employee_gender_idx 2 ON employee (gender) 3 TABLESPACE APPL_IDX;
```
The Bitmap index would then store the two possible values and the binary mapping for each value as it relates to the column data. The bitmap index stores a 1 if the value is true for that row, and a 0 if it is not. Figure 3.16 shows the manner in which a Bitmap index would store the data from the GENDER column.

**FIGURE 3.16** Bitmap index for GENDER column

<table>
<thead>
<tr>
<th>GENDER</th>
<th>ROWID</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>AAAD7fAAJAAAAM8AAA</td>
</tr>
<tr>
<td>F</td>
<td>AAAD7fAAJAAAAM8AB</td>
</tr>
<tr>
<td>M</td>
<td>AAAD7fAAJAAAAM8AC</td>
</tr>
<tr>
<td>F</td>
<td>AAAD7fAAJAAAAM8AAD</td>
</tr>
<tr>
<td>M</td>
<td>AAAD7fAAJAAAAM8AEE</td>
</tr>
<tr>
<td>M</td>
<td>AAAD7fAAJAAAAM8AAF</td>
</tr>
<tr>
<td>M</td>
<td>AAAD7fAAJAAAAM8AAG</td>
</tr>
</tbody>
</table>

Subsequent queries on EMPLOYEE that include an equality match on GENDER in the `WHERE` clause can now locate all row IDs of the rows that satisfy the query with only one I/O. Additionally, queries that use AND or OR conditions on the indexed column in the `WHERE` clause of the SQL statement will be able to do bit-wise row eliminations when determining the query results.

---

**WARNING**

Bitmap indexes should not be used on tables that have high INSERT, UPDATE, or DELETE activity. These DML operations are costly in terms of performance because they cause locking to occur at the bitmap level and require that the entire bitmap for all possible values be rebuilt dynamically. However, each bitmap is only updated once per DML activity, even when several rows are changed. Because of this, bitmapped indexes are best suited for data warehouse and decision support systems.
Consideration should be given to the SORT_AREA_SIZE and PGA_AGREGATE_TARGET init.ora parameters whenever bitmap indexes are used. Increasing the size of each of these parameters generally enhances the speed with which bitmap indexes can be created and manipulated. These three parameters are described in Table 3.21.

### Table 3.21 Important init.ora Parameters Related to Bitmap Indexes

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SORT_AREA_SIZE</td>
<td>Size in bytes of the buffer where sorted bitmap column and row ID information is stored until batch commit</td>
</tr>
<tr>
<td>PGA_AGREGATE_TARGET</td>
<td>Manages the amount of memory, in bytes, assigned to bitmap index creation and bitmap merges following an index range scan</td>
</tr>
</tbody>
</table>

Although not specifically deprecated in Oracle9i, two Oracle8i init.ora parameters designed to manage the creation and merging of bitmap indexes, CREATE_BITMAP_AREA_SIZE and BITMAP_MERGE_AREA_SIZE, are no longer recommended for tuning bitmap activities. The new PGA_AGREGATE_TARGET parameter should be used instead.

### Function-Based Indexes

When a function is applied to an indexed column in an SQL statement, any indexes on that column are negated and therefore not considered during the formulation of the statement’s execution plan. For example, if an index were built on the FIRST_NAME column of the EMPLOYEE, it would not be used by the optimizer in the following SQL statement because a function (UPPER) is being applied to the FIRST_NAME column in the WHERE clause:

```sql
SQL> SELECT last_name, first_name
2  FROM employee
3  WHERE UPPER(first_name) = 'REGINALD';
```

The resulting execution plan would perform a full table scan on the EMPLOYEE table when looking for the employees named Reginald—even
though an index exists on that column. This behavior occurs whenever any function or operation is applied to an indexed column in the SQL statement’s WHERE clause. Function-based indexes resolve this problem by incorporating the function into the index during the index’s creation. Both B-Tree and Bitmapped indexes can be created as function-based. Here is an example of a function-based index built on the FIRST_NAME column of the EMPLOYEE table:

```sql
SQL> CREATE INDEX employee_first_name_upper_idx
2  ON employee (UPPER(first_name));
```

The optimizer may now consider using the new EMPLOYEE_FIRST_NAME_UPPER_IDX index and thus avoid the full table scan when retrieving the employees named Reginald.

Another example of the usefulness of function-based indexes is when you wish to avoid full table scans when using derived columns in a statement’s WHERE clause. For example, in the following query, the column TOTAL_SALE is not an actual column in the table; instead, it is derived by multiplying the UNITS by the PRICE:

```sql
SQL> SELECT product_id, units, price, price*units "TOTAL_SALE"
2  FROM sales
3  WHERE (price*units) > 10000;
```

However, we can create a function-based B-Tree index on the derived TOTAL_SALE column and then use that index to eliminate the full table scan that would otherwise occur in the query:

```sql
SQL> CREATE INDEX sales_total_sale_idx
2  ON sales (price*units)
3  TABLESPACE appl_idx;
```

The optimizer would now be able to consider this index when formulating the execution plan for the previous query.

The init.ora parameter QUERY_REWRITE_ENABLED must be set to TRUE in order to use function-based indexes.

**Reverse Key Indexes**

The **Reverse Key Index (RKI)** is a special type of B-Tree index. The RKI is useful when an index is built on a column that contains sequential numbers. If a traditional B-Tree index is built on a column containing this type of data,
the index tends to develop many levels. Since having more than four levels of depth in a B-Tree can degrade performance, an RKI is well suited to this situation.

Reverse Key Indexes solve this problem by simply reversing the data in the indexed column by reversing the bytes of each column key value and then indexing on the new reversed data, which is generally more evenly distributed across a range of values than the original sequential data was.

For example, if you have determined that an index is needed on the EMP_ID column of the EMPLOYEE table, a Reverse Key index would be appropriate since the data stored in the EMP_ID column is sequential:

```
SQL> SELECT *
2  FROM employee;
```

<table>
<thead>
<tr>
<th>EMP_ID</th>
<th>LAST_NAME</th>
<th>FIRST_NAME</th>
<th>TITLE</th>
<th>DEPT_ID</th>
<th>GENDER</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>Baker</td>
<td>Brenda</td>
<td>Dir Web Development</td>
<td>601</td>
<td>F</td>
</tr>
<tr>
<td>101</td>
<td>Douglass</td>
<td>Laura</td>
<td>Dir of Research</td>
<td>301</td>
<td>F</td>
</tr>
<tr>
<td>102</td>
<td>Jim</td>
<td>Schwarz</td>
<td>Dir of IS</td>
<td>401</td>
<td>M</td>
</tr>
<tr>
<td>110</td>
<td>Block</td>
<td>Robin</td>
<td>Corporate Counsel</td>
<td>501</td>
<td>F</td>
</tr>
<tr>
<td>106</td>
<td>Bron</td>
<td>Reginald</td>
<td>Dir of Marketing</td>
<td>601</td>
<td>M</td>
</tr>
<tr>
<td>108</td>
<td>Weishan</td>
<td>Matt</td>
<td>Technical Writer</td>
<td>601</td>
<td>M</td>
</tr>
<tr>
<td>107</td>
<td>Gastel</td>
<td>Tony</td>
<td>Oracle DBA</td>
<td>401</td>
<td>M</td>
</tr>
</tbody>
</table>

```
SQL> CREATE INDEX employee_emp_id_idx
2  ON employee (emp_id)
3  REVERSE
4  TABLESPACE APPL_IDX;
```

An existing non–Reverse Key index can also be rebuilt into a Reverse Key index using the ALTER INDEX command:

```
ALTER INDEX employee_emp_id_idx REBUILD REVERSE;
```

Figure 3.17 shows an example of how an RKI on the EMP_ID column of the EMPLOYEE table would look internally.

The fact that an index is reverse-keyed is transparent to the application developer and the user. No special syntax is required to use it in your queries.
Reverse Key indexes are only useful for equality and non-equality searches. Queries that perform range scans (e.g., using BETWEEN, >, <) on columns that are Reverse Key indexed will not be able to use the index and will cause full table scans.

**Index Organized Tables**

The performance gain offered by the use of B-Tree, Bitmapped, and Reverse Key indexes results from the fact that the entries in these indexes point directly to the row ID of the corresponding data in the index’s underlying table. This is an effective way to retrieve rows from a table whose rows are not physically stored in any particular order. Tables of this type are referred to as *Heap Tables*. Oracle stores row data in a heap fashion in most tables because the rows are assigned to blocks within the table in a more or less random fashion. This randomness occurs because Oracle does not consider the
contents of a row when deciding where to store it. Instead, Oracle just stores the row in the first block it finds on the table’s Freelist.

If you do wish to store a table’s data in a specific order, then you cannot use a Heap Table. Oracle provides Index Organized Tables (IOTs) just for this purpose. Instead of storing a row ID pointer to where the rest of the row data is stored, the row data is actually stored in its entirety in the index itself. This results in two performance benefits:

- The table rows are stored in index order. If you access the table using its primary key, an IOT will return the rows more quickly than a traditional table.
- The extra I/O experienced by reading the index and then the table when using B-tree indexes is eliminated.

Suppose you have decided to create a new application table called EMPLOYEE_HISTORY that will store employee information by EMP_ID. Since you expect to access this table by EMP_ID, an IOT might be a good choice for creating this table:

```sql
SQL> CREATE TABLE employee_history
2  (employee_id number primary key,
3   last_name varchar2(20),
4   first_name varchar2(20),
5   title varchar2(30),
6   hire_date date,
7   departure_date date)
8  ORGANIZATION INDEX TABLESPACE appl_idx
9  PCTTHRESHOLD 25
10  INCLUDING first_name
11  OVERFLOW TABLESPACE appl_of
12  MAPPING TABLE;
```
All index-organized tables must have a primary key constraint defined on the column that will be the basis of the index. An IOT cannot contain unique constraints or be clustered. Clustering is discussed in a subsequent section.

The important parts of the CREATE TABLE syntax that are related to IOT creation are summarized in Table 3.22.

**TABLE 3.22 Syntax for Index IOT Creation**

<table>
<thead>
<tr>
<th>Clause</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORGANIZATION INDEX</td>
<td>Specifies that this table is an IOT.</td>
</tr>
<tr>
<td>PCTTHRESHOLD</td>
<td>Specifies what percentage of the entire data block to hold open in order to store the row data associated with a primary key value, which must be between 0 and 50 (default 50).</td>
</tr>
<tr>
<td>INCLUDING</td>
<td>Specifies at which column to break a row into two pieces when a row’s length exceeds the size set aside in PCTTHRESHOLD.</td>
</tr>
<tr>
<td>OVERFLOW TABLESPACE</td>
<td>Specifies the tablespace where the second half of the row data will be stored when the row’s length exceeds the size set aside in PCTTHRESHOLD.</td>
</tr>
<tr>
<td>MAPPING TABLE</td>
<td>Causes the creation of an associated mapping table that is needed when creating bitmap indexes on the IOT. Mapping tables are described in more detail at the end of this section.</td>
</tr>
</tbody>
</table>

After populating this table with data, the contents might look something like those shown in Figure 3.18.
FIGURE 3.18 Contents of the EMPLOYEE_HISTORY table

<table>
<thead>
<tr>
<th>EMPLOYEE_ID</th>
<th>LAST_NAME</th>
<th>FIRST_NAME</th>
<th>TITLE</th>
<th>APPL_INDEX_DATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>Baker</td>
<td>Brenda</td>
<td>Dir of Web Dev</td>
<td>07-FEB-98</td>
</tr>
<tr>
<td>101</td>
<td>Douglass</td>
<td>Laura</td>
<td>Dir of Research</td>
<td>10-Jan-97</td>
</tr>
<tr>
<td>102</td>
<td>Schwartz</td>
<td>Jim</td>
<td>Dir of IS</td>
<td>15-MAR-98</td>
</tr>
<tr>
<td>103</td>
<td>Brem</td>
<td>Reginald</td>
<td>Dir of Marketing</td>
<td>29-MAY-96</td>
</tr>
<tr>
<td>107</td>
<td>Gastel</td>
<td>Tony</td>
<td>Oracle DBA</td>
<td>11-JUL-97</td>
</tr>
<tr>
<td>110</td>
<td>Black</td>
<td>Homin</td>
<td>Corporate Counsel</td>
<td>03-JUN-96</td>
</tr>
<tr>
<td>112</td>
<td>Macomber</td>
<td>Bart</td>
<td>Head Honcho Big Cheese Boss Guy and Supreme Leader</td>
<td>03-JUN-96</td>
</tr>
</tbody>
</table>

Figure 3.19 shows how the data for this table would be physically stored between the IOT Segment and the IOT Overflow Segment.

FIGURE 3.19 Physical storage of the EMPLOYEE_HISTORY data

Notice in Figure 3.19 that the entire row of data associated with employee 107, Gastel, fits into one of the blocks allocated to the EMPLOYEE_HISTORY IOT. However, employee 112, Macomber, has a title that is too long to fit into the space reserved by PCTTHRESHOLD. Therefore, the row for Macomber is cut off at the INCLUDING column (FIRST_NAME), and the remaining row data is instead stored in a block belonging to the overflow segment in another tablesapce. Finding the name of this overflow segment is discussed in the following section.

Bitmapped indexes can be built on both heap and index-organized tables. When a Bitmapped index is built on a Heap Table, the index stores bitmaps along with physical row IDs to locate rows in the Heap Table. However, a Bitmap index on an IOT must make use of a mapping table to locate indexed IOT rows. The Mapping Table maps the index’s physical row IDs to
the corresponding logical row IDs used by the index-organized table. IOTs use logical row IDs to manage index access to their rows because the physical row IDs can change as data is added to, or removed from, the IOT. This occurs because the IOT leaf blocks may split as they become full, just as with B-Tree indexes. Therefore, without this mapping table, a leaf block split in the IOT would change the row’s physical row ID and leave the associated Bitmap index entry unusable.

Because of the use of this intermediate mapping table, over time, the Bitmapped index entries may not accurately reflect the true mappings between the logical and physical row IDs in the indexed IOT. This reduces the effectiveness of the index. You can see the extent to which the mapping table is out of sync with the Bitmap index by issuing the following query:

```
SELECT index_name, pct_direct_access
FROM dba_indexes
WHERE pct_direct_access IS NOT NULL;
```

Only IOT tables will have non-null values for PCT_DIRECT_ACCESS. Any index whose PCT_DIRECT_ACCESS value exceeds 30 percent should have its Bitmapped index rebuilt in order to maintain its effectiveness.

---

There is only one mapping table to maintain per IOT, even if there are multiple Bitmap indexes on that IOT.

---

There is an Oracle9i documentation bug that states that the following command can be used to rebuild a mapping table on the EMPLOYEE_HISTORY IOT:

```
ALTER TABLE employee_history MAPPING TABLE UPDATE BLOCK REFERENCES;
```

However, as of this writing, this command is not available.

### Index Data Dictionary Views

The data dictionary views DBA_INDEXES, DBA_SEGMENTS, and DBA_TABLES all contain information about one or more of the index types you have seen so far.
**DBA_INDEXES**

DBA_INDEXES is the data dictionary view that contains most of the information a DBA needs regarding indexes. Table 3.23 describes the columns of the DBA_INDEXES view that are related to the indexes discussed in the previous sections.

**Table 3.23** Description of Selected DBA_INDEXES Columns

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INDEX_TYPE</td>
<td>Specifies what type the index is:</td>
</tr>
<tr>
<td></td>
<td>NORMAL (Standard B-Tree)</td>
</tr>
<tr>
<td></td>
<td>NORMAL/REV (Reverse Key)</td>
</tr>
<tr>
<td></td>
<td>FUNCTION-BASED NORMAL</td>
</tr>
<tr>
<td></td>
<td>BITMAP</td>
</tr>
<tr>
<td></td>
<td>IOT—TOP (Index Organized Table)</td>
</tr>
<tr>
<td></td>
<td>CLUSTER</td>
</tr>
<tr>
<td></td>
<td>LOB (Large Object)</td>
</tr>
<tr>
<td>COMPRESSION</td>
<td>Indicates whether the index is compressed. Two values are used:</td>
</tr>
<tr>
<td></td>
<td>DISABLED</td>
</tr>
<tr>
<td></td>
<td>ENABLED</td>
</tr>
<tr>
<td>FUNCIDX_STATUS</td>
<td>Indicates the status of the function-based index:</td>
</tr>
<tr>
<td></td>
<td>NULL (not a function-based index)</td>
</tr>
<tr>
<td></td>
<td>ENABLED</td>
</tr>
<tr>
<td></td>
<td>DISABLED</td>
</tr>
</tbody>
</table>

A sample query against DBA_INDEXES showing index information on the indexes we discussed is shown in Figure 3.20.

**Figure 3.20** Sample query showing index types

```sql
SQL> SELECT index_name, index_type, table_name, compression, funcidx_status
FROM dba_indexes
WHERE table_name IN ('EMPLOYEE_HISTORY', 'EMPLOYEE')
ORDER BY index_name;

+---------------+------------------+------------------+------------------+-------------------+
<table>
<thead>
<tr>
<th>INDEX_NAME</th>
<th>INDEX_TYPE</th>
<th>TABLE_NAME</th>
<th>COMPRESSION</th>
<th>FUNCIDX_STATUS</th>
</tr>
</thead>
<tbody>
<tr>
<td>EMPLOYEE_EMP_ID_IDX</td>
<td>NORMAL/REV</td>
<td>EMPLOYEE</td>
<td>DISABLED</td>
<td>ENABLED</td>
</tr>
<tr>
<td>EMPLOYEE_FIRST_NAME_IDX</td>
<td>FUNCTION-BASED NORMAL</td>
<td>EMPLOYEE</td>
<td>DISABLED</td>
<td>ENABLED enabled</td>
</tr>
<tr>
<td>EMPLOYEE_GENDER_IDX</td>
<td>BITMAP</td>
<td>EMPLOYEE</td>
<td>DISABLED</td>
<td>ENABLED</td>
</tr>
<tr>
<td>EMPLOYEE_HIST_LAST_NAME_IDX</td>
<td>NORMAL</td>
<td>EMPLOYEE_HISTORY</td>
<td>DISABLED</td>
<td>ENABLED</td>
</tr>
<tr>
<td>EMPLOYEE_LAST_NAME_IDX</td>
<td>NORMAL</td>
<td>EMPLOYEE_HISTORY</td>
<td>DISABLED</td>
<td>ENABLED</td>
</tr>
<tr>
<td>SYS_IOT_B771</td>
<td>IOT — TOP</td>
<td>EMPLOYEE_HISTORY</td>
<td>DISABLED</td>
<td>DISABLED</td>
</tr>
</tbody>
</table>
```
**DBA_SEGMENTS**

The only information lacking in the query in Figure 3.19 is the specifics of the overflow segment for the IOT, EMPLOYEE_HISTORY. This information is available from DBA_SEGMENTS, as shown below.

```sql
SQL> SELECT segment_name,
        2   segment_type,
        3   tablespace_name
        4  FROM dba_segments
        5  WHERE segment_name LIKE '%IOT%';
```

<table>
<thead>
<tr>
<th>SEGMENT_NAME</th>
<th>SEGMENT_TYPE</th>
<th>TABLESPACE_NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYS_IOT_OVER_16102</td>
<td>TABLE</td>
<td>APPL_OF</td>
</tr>
<tr>
<td>SYS_IOT_TOP_16102</td>
<td>INDEX</td>
<td>APPL_IDX</td>
</tr>
</tbody>
</table>

This query shows the two components of the IOT EMPLOYEE_HISTORY: the index itself, SYS_IOT_TOP_16102, and the overflow segment SYS_IOT_OVER_16102. The numeric portion of the names was system generated at the creation of the IOT.

**DBA_TABLES**

In order to more clearly see the relationship between the EMPLOYEE_HISTORY IOT and its associated overflow segment, we can query DBA_TABLES as shown below.

```sql
SQL> SELECT table_name,
        2   iot_name,
        3   iot_type,
        4   tablespace_name
        5  FROM dba_tables
        6  WHERE table_name = 'EMPLOYEE_HISTORY'
        7  OR iot_name = 'EMPLOYEE_HISTORY';
```

<table>
<thead>
<tr>
<th>TABLE_NAME</th>
<th>IOT_NAME</th>
<th>IOT_TYPE</th>
<th>TABLESPACE</th>
</tr>
</thead>
<tbody>
<tr>
<td>EMPLOYEE_HISTORY</td>
<td>IOT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SYS_IOT_OVER_16102</td>
<td>EMPLOYEE_HISTORY IOT_OVERFLOW</td>
<td>APPL_OF</td>
<td></td>
</tr>
</tbody>
</table>

Note that the table SYS_IOT_OVER_16102 has the entry EMPLOYEE_HISTORY as its associated table. Further, observe that the entry for TABLESPACE_NAME for the EMPLOYEE_HISTORY table is NULL since the actual segment that stores the data is an index, not a table.
Improving Application Performance

**Partitions**

Another technique for reducing the amount of I/O during query execution is to divide the table data into smaller subsets, called *partitions*. Once properly partitioned, indexed, and analyzed, large table queries will benefit from the CBO’s ability to use partition elimination in order to skip table partitions that do not contain relevant data. In addition, many table maintenance

---

**Oracle Objective**

Monitor indexes to determine usage

<table>
<thead>
<tr>
<th>Identifying Unused Indexes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Depending on the execution plans the optimizer chooses, some application indexes may never be utilized while application users are performing their work. If an index is not being used, then dropping it may improve performance by eliminating the overhead of the index maintenance when DML is performed on the indexed table. Dropping unused indexes will also free up tablespace for use by other indexes. Oracle9i introduces a new data dictionary view, <code>V$OBJECT_USAGE</code>, that can be used to identify unused indexes. The technique for monitoring index usage is briefly described here:</td>
</tr>
</tbody>
</table>
| You can use the following command to activate monitoring on an index called `EMP_LAST_NAME_IDX`:

```
ALTER INDEX hr.emp_last_name_idx MONITORING USAGE;
```

After a period of normal user activity, you can disable the monitoring using the following command:

```
ALTER INDEX hr.emp_last_name_idx NOMONITORING USAGE;
```

You can then query the `INDEX_NAME` and `USED` column in the `V$OBJECT_USAGE` dynamic performance view to determine whether the index was accessed during the monitoring period. A `YES` value in the `USED` column indicates that the index was accessed at least once during the monitoring period. A value of `NO` in the `USED` column indicates that the index was not accessed during the monitoring period and may be a candidate for dropping. |

---

**Partitions**

Another technique for reducing the amount of I/O during query execution is to divide the table data into smaller subsets, called *partitions*. Once properly partitioned, indexed, and analyzed, large table queries will benefit from the CBO’s ability to use partition elimination in order to skip table partitions that do not contain relevant data. In addition, many table maintenance
operations can also be performed at the partition level. Oracle9i offers four methods of partitioning a table.

**Oracle Objective**

Describe partitioning methods

### Oracle Table Partition Methods

Before a table can be partitioned, a suitable partition method must be selected. The choice of a partition method will often be driven by the data stored in the partition key—the column or columns you wish to use as the basis for subdividing the table’s data. For example, suppose you are the DBA at a large university. As part of a larger data warehousing initiative, the university’s data architect has asked you to build a large table (i.e., STUDENT_HISTORY) that will be used to store information about the graduates of the institution’s academic programs over a five-year period. Each year a new set of records will be added for that year’s graduates, and the oldest records (those from the graduates of five years prior) will be purged from the table. For simplicity, assume the table will be composed of the four columns shown in Table 3.24.

| **TABLE 3.24** Description of STUDENT_HISTORY Table |
|---------------------------------|-------------------------------------------------|
| **Column Name**                | **Description**                                 |
| STUDENT_ID                     | Student’s identifying number                    |
| DEGREE                         | The three character abbreviation for the degree the student earned: BS, BA, BBA, BFA, MA, MBA, MFA, MS, PHD |
| GRADUATION_DATE                | The date the student graduated from the university |
| FINAL_GPA                      | The student’s final grade point average upon graduation. |

This sample table structure will be used for demonstrating the four types of partitions:

- Range partitions
- List partitions
- Hash partitions
- Composite partitions

**RANGE PARTITIONING**

*Range partitioning* represents one option for subdividing contents of the STUDENT_HISTORY table. Range partitioning uses ranges of column values to determine into which partition a row of data will be inserted. Suppose you’ve decided to use Range partitioning on the GRADUATION_DATE column to partition the STUDENT_HISTORY table. This means that GRADUATION_DATE is the partition key:

```sql
SQL> CREATE TABLE student_history
2 (student_id NUMBER(10),
3  degree VARCHAR2(3),
4  graduation_date DATE,
5  final_gpa NUMBER)
6  PARTITION BY RANGE (graduation_date)
7 (PARTITION p_1997 VALUES LESS THAN
8   (TO_DATE('01-JUN-1997','DD-MON-YYYY')) TABLESPACE hist_tab01,
9  PARTITION p_1998 VALUES LESS THAN
10  (TO_DATE('01-JUN-1998','DD-MON-YYYY')) TABLESPACE hist_tab02,
11  PARTITION p_1999 VALUES LESS THAN
12  (TO_DATE('01-JUN-1999','DD-MON-YYYY')) TABLESPACE hist_tab03,
13  PARTITION p_2000 VALUES LESS THAN
14  (TO_DATE('01-JUN-2000','DD-MON-YYYY')) TABLESPACE hist_tab04,
15  PARTITION p_2001 VALUES LESS THAN
16  (TO_DATE('01-JUN-2001','DD-MON-YYYY')) TABLESPACE hist_tab05,
17  PARTITION p_ERROR VALUES LESS THAN
18  (MAXVALUE) TABLESPACE hist_tab06);
```

The above syntax would create a table with six partitions, named P_1997 through P_2001, and P_ERROR. For performance reasons, each of these partitions is physically stored in a separate tablespace (HIST_TAB01 through HIST_TAB06). Every row inserted into the table will be placed into the appropriate partition, based on the student’s graduation date. Figure 3.21 illustrates this concept.
FIGURE 3.21 Range-partitioned STUDENT_HISTORY table

When examining the above structure, several rules related to Range partitioning should be considered:

- The partition key can be composed of up to 16 columns.

- A table can have up to 65,636 partitions.

- There cannot be any gaps in the partition ranges. For example, you could not have partitions for graduation dates of 1991–1993 and 1995–2001 (skipping 1994).

- All rows stored in a partition will have values less than, and not equal to the upper bound for that partition. In other words, a record inserted for student who graduated on 01-JUN-2000 will be stored in the P_2001 partition, not the P_2000 partition. Therefore, insertion
of a student record with a graduation date of 01-JUN-2001 or later will cause that record to be stored in the P_ERROR partition.

- The value specified in the LESS THAN clause must be a literal, a date, or a constant padded by the RPAD function. Date values must also use the TO_DATE function to include the century.

- Range-partitioned tables cannot contain columns with LONG or LONG RAW datatypes.

- Updates that would cause a record to move across partition boundaries are not allowed unless the ENABLE ROW MOVEMENT clause is specified at table creation. Without this clause, attempting to update a student’s graduation date from 16-MAY-95 to 16-MAY-96 would require that the record be moved from partition P_1995 to P_1996 and would not be allowed.

- Attempts to insert records that do not fit into any defined partition ranges will cause an ORA-14400: Insert partition key is beyond highest legal partition key error. If the P_ERROR partition with the MAXVALUE clause had not been specified, the insertion of a student record with a graduation date of 01-JUN-2001 or later, or value of NULL, would not be allowed.

The selection of GRADUATION_DATE as the table’s partition key is probably ineffective if the number of students who graduate varies considerably each year. If this is the case, then the number of rows stored within each partition will become uneven, with some partitions growing much larger than other partitions. While not always detrimental to performance, this unevenness can cause parallel operations like Parallel Query and Parallel DML to perform poorly. If this is the case, then Hash partitioning could be considered in place of Range partitioning.

Parallel operations are described in greater detail in Chapter 10, “Operating System Tuning.”

LIST PARTITIONING

List partitions are similar to Range partitions except they are based on a set of specified values rather than a range of values. For example, suppose
you've decided to use list partitioning on the DEGREE column to partition the STUDENT_HISTORY table. This means that DEGREE is the partition key:

```sql
SQL> CREATE TABLE student_history
2 (student_id NUMBER(10),
3  degree VARCHAR2(3),
4  graduation_date DATE,
5  final_gpa NUMBER)
6  PARTITION BY LIST (degree)
7  (PARTITION p_undergrad VALUES ('BS','BA','BBA','BFA')
8    TABLESPACE hist_tab01,
9  PARTITION p_graduate VALUES ('MA','MBA','MFA','MS')
10    TABLESPACE hist_tab02,
11  PARTITION p_doctorate VALUES ('PHD')
12    TABLESPACE hist_tab03);
```

The above syntax would create a table with three partitions, named P_UNDERGRAD, P_GRADUATE, and P_DOCTORATE. For performance reasons, each of these partitions is physically stored in a separate tablespace (HIST_TAB01 through HIST_TAB03). Every row inserted into the table will be placed into the appropriate partition, based on the student's degree code. Figure 3.22 illustrates this concept.

**Figure 3.22** List-partitioned STUDENT_HISTORY table

```
INSERT INTO
student_history
(student_id, degree,
graduation_date, final_gpa)
VALUES
(123456789, 'BBA', '12-MAY-99', 3.75);
```
When examining the above structure, one rule related to list partitioning should be considered: Values that do not meet any of the conditions of the partition key will raise an ORA-14400 error. Unlike Range partitioning, Oracle9i Release 1 currently does not support use of the MAXVALUE option to eliminate this problem.

Specific physical segment attributes, such as STORAGE parameters, can be assigned at the partition level. These attributes will override the table’s default attributes.

**HASH PARTITIONING**

*Hash partitions* use a hashing algorithm to assign inserted records into partitions. This has the effect of maintaining a relatively even number of records in each partition, thus improving the performance of parallel operations like Parallel Query and Parallel DML. Suppose you’ve decided to use hash partitioning on the STUDENT_ID column instead of Range partitioning to partition the STUDENT_HISTORY table. This means that STUDENT_ID is the partition key:

```sql
SQL> CREATE TABLE student_history
2 (student_id      NUMBER(10),
3  degree          VARCHAR2(3),
4  graduation_date DATE,
5  final_gpa       NUMBER)
6  PARTITION BY HASH (student_id)
7  PARTITIONS 6
8  STORE IN (hist_tab01, hist_tab02, hist_tab03,
9           hist_tab04, hist_tab05, hist_tab06);
```

The above syntax would create a table with six partitions, with system-generated names. Whenever a new record is inserted into the STUDENT_HISTORY table, Oracle will pass the value in STUDENT_ID through the hashing algorithm, and then use the resulting hashed value to determine into which partition the row should be inserted. Figure 3.23 illustrates this concept.
When examining the above structure, several rules related to hash partitioning should be considered:

- The partition key should have high cardinality—the column should contain unique values or very few duplicate values.
- Hash partitions work best when applications retrieve the data from the partitioned table via the unique key. Range lookups on hash partitioned tables derive no benefit from the partitioning.
- When they are properly indexed, hash partitioned tables improve the performance of joins between two or more partitioned tables.
- Updates that would cause a record to move across partition boundaries are not allowed.
- Local hash indexes can be built on each partition.
Indexing strategies for partitioned tables is discussed later in this section.

Oracle recommends that the total number of partitions be a power of two in order to maximize the effectiveness of the hash partition algorithm.

**COMPOSITE PARTITIONING**

As the name implies, *composite partitioning* represents a combination of both range and hash partitioning. This type of partition is useful when Range partitioning is desired, but when the resulting number of values within each range would be uneven. Composite partitioning creates Range partitions that are in turn subdivided into hashed subpartitions:

```sql
SQL> CREATE TABLE student_history
  2 (student_id NUMBER(10),
  3  degree VARCHAR2(3),
  4  graduation_date DATE,
  5  final_gpa NUMBER)
  6  PARTITION BY RANGE (graduation_date)
  7  SUBPARTITION BY HASH(student_id) SUBPARTITIONS 4
  8  STORE IN (hist_tab01, hist_tab02, hist_tab03, hist_tab04)
  9  (PARTITION p_1997 VALUES LESS THAN
 10   (TO_DATE('01-JUN-1997','DD-MON-YYYY')),
 11  PARTITION p_1998 VALUES LESS THAN
 12   (TO_DATE('01-JUN-1998','DD-MON-YYYY')),
 13  PARTITION p_1999 VALUES LESS THAN
 14   (TO_DATE('01-JUN-1999','DD-MON-YYYY')),
 15  PARTITION p_2000 VALUES LESS THAN
 16   (TO_DATE('01-JUN-2000','DD-MON-YYYY')),
 17  PARTITION p_2001 VALUES LESS THAN
 18   (TO_DATE('01-JUN-2001','DD-MON-YYYY')),
 19  PARTITION p_ERROR VALUES LESS THAN (MAXVALUE));
```

The above syntax would create a table with six Range partitions, named P_1997 through P_2001, and P_ERROR. Each of these Range partitions would be divided into four hash subpartitions, each with a system-generated partition name. Figure 3.24 illustrates this concept.
When examining the above structure, several guidelines related to composite partitioning should be considered:

- The partitions are logical structures only; the table data is physically stored at the sub-partition level.
- Composite partitions are useful for historical, date-related queries at the partition level.
- Composite partitions are also useful for parallel operations (both query and DML) at the subpartition level.
- Partition-wise joins are also supported through the use of composite local indexes. Global local indexes are not supported.

The difference between local and global partitioned indexes is discussed later in the next section.
Indexing Partitioned Tables

After a range, list, hash, or composite partitioned table has been created, it should be indexed in order to take maximum advantage of the performance enhancing benefits of partitioned tables. Proper indexing of partitioned tables will also improve the manageability of the indexes when performing routine index maintenance tasks described earlier in this chapter. There are two methods of classifying partitioned indexes. The first category, local versus global indexes, is related to whether the index’s partitioning structure matches that of the underlying, indexed table.

**Local Partition Indexes**  Partitioned indexes are said to be local when the number of partitions in the index match the underlying table’s partitions on a one-to-one basis. Therefore, if a table contains four partitions, the local index on that table will also contain four partitions, each indexing one of the four table partitions. Figure 3.25 illustrates the relationship between a local partitioned index and its associated partitioned table.

When examining the above structure, several guidelines related to local partition indexes should be considered:

- Local partitioned indexes can use any of the four partition types.
- Oracle automatically maintains the relationship between the table’s partitions and the index’s partitions. If a table partition is merged, split, or dropped, the associated index partition will also be merged, split, or dropped.
- Any bitmapped index built on a partitioned table must be a local index.

**Figure 3.25** Relationship between local partitioned indexes and tables
Global Partition Indexes  Partitioned indexes are said to be global when the number of partitions in the index do not match the underlying table’s partitions on a one-to-one basis. For example, if a table contains four partitions, its global index may be comprised of only two partitions, each of which may encompass one or more of the table partitions. Figure 3.26 illustrates the relationship between a global partitioned index and its associated partitioned table.

**FIGURE 3.26** Relationship between global partitioned indexes and tables

When examining the above structure, several guidelines related to global partition indexes should be considered:

- Although global indexes can be built on a range, list, hash, or composite partitioned table, the global partitioned index itself must be Range partitioned.

- The highest partition of a global index must be defined using the `MAXVALUE` parameter.

- Performing maintenance operations on partitioned tables can cause their associated global indexes to be placed into an invalid state. Indexes in an invalid state must be rebuilt before users can access them.

To avoid inadvertent index invalidation, Oracle recommends the use of local indexes over global indexes whenever possible.
• Creating a global partitioned index with the same number of partitions as its underlying table (i.e. simulating a local partitioned index) does not substitute for a local partitioned index. The indexes would not be considered the same by the CBO despite their similar structures.

The second method of classifying partitioned indexes, prefixed versus non-prefixed indexes, is related to whether the index contains the partition key and where the partition key appears within the index structure.

**Prefixed Partition Indexes**  Partioned indexes are said to be prefixed whenever the left-most column of the index is the same as the underlying table’s partition key column. For example, if the `STUDENT_HISTORY` table were partitioned on the `GRADUATION_DATE` column, a prefixed partitioned index would also use `GRADUATION_DATE` as the first column in its index definition.

| Note | Prefixed indexes can be either unique or non-unique. |

**Non-prefixed Partitioned Indexes**  Partitioned indexes are said to be non-prefixed when the left-most column of the index is not the same as the underlying table’s partition key column. For example, if the `STUDENT_HISTORY` table were partitioned on the `GRADUATION_DATE` column, a non-prefixed partitioned index would use some column other than `GRADUATION_DATE` as the first column in its index definition.

| Note | Non-prefixed can be unique or nonunique. However, unique non-prefixed indexes are only allowed when the index’s partition key is a subset of the index key. |

The two classifications of partitioned indexes can be combined in several ways. Figure 3.27 shows the possible combinations of index classifications.

**FIGURE 3.27** Combinations of partitioned indexes

| |
|---|---|---|
| Prefixed index | Local index | Global index |
| | Local, prefixed partitioned index | Global, prefixed partitioned index |
| Non-prefixed index | Local, non-prefixed partitioned index | Not Allowed |
The syntax for creating each of the index types in Figure 3.27 is shown below.

**Creating a Local, Prefixed Partition Index**  If a local, prefixed partitioned index was to be built on the Range partitioned STUDENT_HISTORY table from the previous example, the following syntax could be used:

```
SQL> CREATE INDEX student_history_lp_idx
ON student_history (graduation_date) LOCAL;
```

The above syntax would create a six-partition index (the same number of partitions as the underlying table) whose leading column would be **GRADUATION_DATE** (the same column that makes up the underlying table’s partition key).

**Creating a Local, Non-prefixed Partition Index**  If a local, non-prefixed partitioned index was to be built on the **DEGREE** column of the Range partitioned STUDENT_HISTORY table, the following syntax could be used:

```
SQL> CREATE INDEX student_history_lnp_idx
ON student_history (final_gpa) LOCAL;
```

The above syntax would create a six-partition index (the same number of partitions as the underlying table) whose leading column would be **FINAL_GPA** (a column different from that which makes up the underlying table’s partition key).

**Creating a Global, Prefixed Partition Index**  If a global, prefixed partitioned index was to be built on the **GRADUATION_DATE** column of the Range partitioned STUDENT_HISTORY table, the following syntax could be used:

```
SQL> CREATE INDEX student_history_gp_idx
ON student_history (graduation_date)
GLOBAL PARTITION BY RANGE (graduation_date)
(PARTITION p_199n VALUES LESS THAN
(TO_DATE('01-JUN-1999','DD-MON-YYYY')),
PARTITION p_200n VALUES LESS THAN
(TO_DATE('01-JUN-2009','DD-MON-YYYY')),
PARTITION p_error VALUES LESS THAN (MAXVALUE));
```

The above syntax would create a three-partition index whose leading column would be **GRADUATION_DATE**.
In addition to the above partitioned index types, traditional non-partitioned B-Tree indexes can also be built on partitioned tables. These indexes are referred to as global, prefixed non-partitioned indexes.

**How the Partitioned Tables Affect the Optimizer**

Once partitioned tables and indexes have been created, it is up to the cost-based optimizer to decide how to use them effectively when executing application queries. Because the Oracle9i CBO is very “partition aware,” it can eliminate partitions that will not be part of the query result. For example, suppose the following query were performed on the Range partitioned STUDENT_HISTORY table:

```sql
SQL> SELECT *
  2  FROM student_history
  3  WHERE graduation_date = '16-MAY-97';
```

Because of the presence of the table and index partitions, the CBO will know that only one partition (P_1997) will need to be accessed to satisfy the query. No other partitions will even need to be considered. This quick elimination of unneeded partitions to improve query performance is referred to as **partition pruning**.

The CBO cannot prune table partitions if the query applies a function to the partition key column. The CBO cannot prune index partitions if the query applies a function to the partition key column on a non-function based index.

Partitioned tables and indexes also help the CBO develop effective execution plans by influencing the manner in which joins are performed between two or more partitioned tables. When performed in parallel using Oracle Parallel Query, there are two possible join methods that are considered by the CBO: full partition-wise joins and partial partition-wise joins. Partition-wise joins occur any time two tables are joined on their partition key columns. Partition-wise joins can be performed either serially or in parallel. Partial partition-wise joins occur any time two tables are joined on the partition key columns of one of the two tables. Partial partition-wise joins can only be performed in parallel. Each of these join types reduces the overall CPU and memory utilized to service a query, particularly on very large partitioned tables.
As described in the previous section, accurate table and index statistics are vital for efficient CBO performance. The GATHER_TABLE_STATS procedure in the DBMS_STATS package can be used to gather global statistics on partitioned tables. These statistics can be gathered at the segment, partition, or subpartition level. The following example shows how statistics can be gathered on the P_1998 partition of the STUDENT_HISTORY table:

```sql
SQL> EXECUTE DBMS_STATS.GATHER_TABLE_STATS (
>  ownname => 'APPS',
>  tabname => 'STUDENT_HISTORY',
>  partname => 'P_1998',
>  granularity => 'PARTITION');
```

The GATHER_INDEX_STATS procedure in the DBMS_STATS package can be used to gather global statistics on partitioned indexes. These statistics can be gathered at the segment, partition, or subpartition level. The following example shows how statistics can be gathered on the P_200n partition of the STUDENT_HISTORY_GP_IDX index:

```sql
SQL> EXECUTE DBMS_STATS.GATHER_INDEX_STATS (
>  ownname => 'SYS',
>  indname => 'STUDENT_HISTORY_GP_IDX',
>  partname => 'P_200n');
```

The GATHER_SCHEMA_STATS and GATHER_DATABASE_STATS procedures in the DBMS_STATS package can also be used to gather statistics for partitioned tables and indexes.

### Data Dictionary Views Related to Partitions

There are 15 data dictionary views that contain information related to partitioned tables and indexes. The names and descriptions of these views are shown in Table 3.25.

<table>
<thead>
<tr>
<th>View Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBA_IND_PARTITIONS</td>
<td>Contains partition details about index partitions</td>
</tr>
<tr>
<td>DBA_IND_SUBPARTITIONS</td>
<td>Contains partition details about index subpartitions</td>
</tr>
<tr>
<td>View Name</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>DBA_LOB_PARTITIONS</td>
<td>Contains partition details about large object partitions</td>
</tr>
<tr>
<td>DBA_LOB_SUBPARTITIONS</td>
<td>Contains partition details about large object subpartitions</td>
</tr>
<tr>
<td>DBA_PART_COL_STATISTICS</td>
<td>Contains columns statistics for partitioned tables</td>
</tr>
<tr>
<td>DBA_PART_HISTOGRAMS</td>
<td>Contains histogram statistics for partitioned tables</td>
</tr>
<tr>
<td>DBA_PART_INDEXES</td>
<td>Contains details about each partitioned index</td>
</tr>
<tr>
<td>DBA_PART_KEY_COLUMNS</td>
<td>Contains the partition key columns for all partitioned tables and indexes</td>
</tr>
<tr>
<td>DBA_PART_LOBS</td>
<td>Contains details about each partitioned large object</td>
</tr>
<tr>
<td>DBA_PART_TABLES</td>
<td>Contains details about each partitioned table</td>
</tr>
<tr>
<td>DBA_SUBPART_COL_STATISTICS</td>
<td>Contains subpartition column statistics</td>
</tr>
<tr>
<td>DBA_SUBPART_HISTOGRAMS</td>
<td>Contains subpartition histogram statistics</td>
</tr>
<tr>
<td>DBA_SUBPART_KEY_COLUMNS</td>
<td>Contains the partition key column for each subpartitioned table or index</td>
</tr>
<tr>
<td>DBA_TAB_PARTITIONS</td>
<td>Contains details about each table’s individual partitions</td>
</tr>
<tr>
<td>DBA_TAB_SUBPARTITIONS</td>
<td>Contains details about each table’s individual subpartitions</td>
</tr>
</tbody>
</table>
Index and Hash Clusters

Indexes improve the performance of queries because they eliminate full table scans by directing Oracle to the precise location on the requested rows. This reduces the total number of data blocks that must be read in order to satisfy a query. A second mechanism for improving performance by minimizing I/O can be effected through the use of clusters. A cluster is a group of one or more tables whose data is stored together in the same data blocks. In this way, queries that utilize these tables via a table join don’t have to read two sets of data blocks to get their results; they only have to read one. There are two types of clusters available in Oracle9i: the Index cluster and the Hash cluster.

Index Clusters

Index clusters are used to store the data from one or more tables in the same physical Oracle blocks. The usefulness of clusters is limited to very specific situations. In general, clustered tables should have these attributes:

- Always be queried together and only infrequently on their own
- Have little or no DML activity performed on them after the initial load
- Have roughly equal numbers of child records for each parent key

If you have tables that meet these criteria, you may want to consider creating an Index cluster to store the data for these tables. Clustered tables behave just like unclustered tables, but their storage space is taken, not from a tablespace, but from a cluster, which itself is a segment stored in a tablespace. The cluster is accessed via an index that contains entries that point to the specific key values on which the tables are joined (i.e., clustered).

Suppose you have two tables, TEACHER and STUDENT, which contain the following data:

```sql
SQL> SELECT * FROM teacher;
```
You have noticed that the application rarely retrieves teacher information without the accompanying student information. If the number of students per teacher is uniform, you may decide to cluster the data from these two tables into the same physical blocks to improve query performance. The following example shows the SQL statements required to create a cluster called TEACHER_STUDENT, an index on the cluster called TEACHER_STUDENT_IDX, and tables TEACHER and STUDENT in the cluster:

```sql
SQL> CREATE CLUSTER teacher_student
    2  (teacher_id number)
    3  SIZE 1000
    4  STORAGE (initial 500K next 500K pctincrease 0);
    5  TABLESPACE apps_clst;
```

The above command creates a cluster called TEACHER_STUDENT and specifies that the column TEACHER_ID will be the cluster key or column around which the two tables' data will be clustered. This column was chosen because it will be the column that the TEACHER and STUDENT tables will have in common. The parameter SIZE specifies how many cluster keys we expect to have
per Oracle block. If we have a 2048-byte (2K) Oracle block size, then this parameter indicates that we expect to get about two (2048/1000 = 2.048) cluster keys per block. In other words, you expect to store the data for two teachers and all of their students in each Oracle block. Choosing the proper value for SIZE is difficult if all the cluster key values (teachers) don’t have the same number of clustered records (students). However, setting this value properly is critical to gaining optimal cluster performance:

```sql
SQL> CREATE INDEX TEACHER_STUDENT_IDX
  2  ON CLUSTER teacher_student
  3  TABLESPACE apps_idx;
Index created.
```

Notice that the indexed column is not specified when creating an index on a cluster segment. The index is automatically created on the cluster key column, TEACHER_ID.

Now that the cluster structure exists, you can create the TEACHER and STUDENT tables in the cluster:

```sql
SQL> CREATE TABLE TEACHER
  2  (teacher_id number,
  3   last_name varchar2(20),
  4   first_name varchar2(20),
  5   room_number number)
  6  CLUSTER teacher_student
  7  (teacher_id);
Table created.
```

```sql
SQL> CREATE TABLE student
  2  (student_id number,
  3   last_name varchar2(20),
  4   first_name varchar2(20),
  5   teacher_id number)
  6  CLUSTER teacher_student
  7  (teacher_id);
Table created.
```

Note that no tablespace specification is given when creating the tables in the cluster. This is not because we want to use the default tablespace for the user, but because the cluster itself is the storage area for each table’s data. In
this way, any data inserted into the TEACHER or STUDENT table will be stored in the same physical blocks—the blocks of the cluster segment they are contained in. Figure 3.28 shows the relationship between two clustered tables, the cluster itself, the cluster index, and the tablespaces they are both stored in.

**FIGURE 3.28** Index cluster and table relationship

### Hash Clusters

*Hash clusters* are used in place of a traditional index to quickly find rows stored in a table. Like cluster indexes, the usefulness of Hash clusters is also limited to very specific situations. In general, Hash cluster tables should have these attributes:

- Have little or no DML activity performed on them after the initial load
- Have a uniform distribution of values in the indexed column
- Have a predictable number of values in the indexed column
- Be queried with SQL statements that utilize equality matches against the indexed column in their `WHERE` clauses

Rather than examine an index and then retrieve the rows from a table, a Hash cluster’s data is stored so queries against the cluster can use a hashing algorithm to determine directly where a row is stored with no intervening index required. In order to compare a Hash cluster and an index on a non–Hash cluster table, assume you issue the following query on the EMPLOYEE table:

```
SQL> SELECT * 
    2   FROM employee 
    3   WHERE emp_id = 100;
```
If the EMPLOYEE table were created as a non–Hash cluster table, a query like this would result in the following operations:

- Find the EMP_ID in the index on the EMP_ID column.
- Use the row ID value stored in the index to locate the row in the table.
- Read the row from the table, and return the information to the user.

This requires at least two I/Os to retrieve the row information, one (or more) on the index and one on the table. If, however, this table were created using a Hash cluster, then the query would involve these operations instead:

- The value in the WHERE clause (100) is passed through the same hashing algorithm that is used when inserting rows. This process occurs in memory and requires no I/O. The resulting value produced by the algorithm points directly to the location of the row in the table.
- The row is read, and the information is returned to the user.

In this case, only one I/O was required to retrieve the row, instead of the two or more with the traditional table/B-Tree index combination.

**Comparing Physical Storage by Table Type**

Now that you know of all the options available to the DBA for storing and accessing data, here is a brief recap comparing each of the physical table types with regard to how they actually store their rows. The list is from the least-ordered to the most-ordered structure:

- **Heap Tables**: Data is stored in the order in which the table’s blocks are filled. As blocks come on and off the table’s Freelist, the order in which the rows are stored becomes essentially random.
- **Cluster Index**: Related rows are stored together in related groupings, around the cluster key value. Between groupings there is no guarantee of the order of the data.
- **Index-organized Table**: Data is stored strictly in index order.

Careful testing should be performed before implementing Index clusters and Hash clusters. Trying to apply them when the table data is not appropriate for their usage, or specifying their storage parameters incorrectly at cluster creation, can actually hinder performance instead of improve it.
OLTP vs. DSS Tuning Requirements

So far, you have examined the indexing techniques available to you for improving query execution. Now it is time to consider how the appropriateness of those options differs when considering the two primary types of database systems in use today: Online Transaction Processing Systems (OLTP) and Decision Support Systems (DSS).

### Tuning OLTP Systems

*Online Transaction Processing* (OLTP) systems tend to be accessed by large numbers of users doing short DML transactions. Order taking, inventory management, and scheduling systems are all examples of OLTP systems. Users of OLTP systems are primarily concerned with throughput: the total time it takes to place an order, remove an item from inventory, or schedule an appointment.

OLTP systems need enough B-Tree and Reverse Key indexes to meet performance goals but not so many as to slow down the performance of INSERT, UPDATE, and DELETE activity. Bitmap indexes are not a good choice for OLTP systems because of the locking issues they can cause.

Table and index statistics should be gathered regularly if the CBO is used because data volumes tend to change quickly in OLTP systems. OLTP indexes should also be rebuilt frequently if the data in the tables they are indexing is frequently modified.

### Tuning DSS Systems

*Decision Support Systems* (DSS) and data warehouses represent the other end of the tuning spectrum. These systems tend to have very little if any INSERT, UPDATE, and DELETE activity, except when data is mass loaded or purged at the end of the week, month, quarter, or year. Users of these systems are concerned with response time, which is the time it takes to get the results from their queries.

DSS makes heavy use of full table scans so the appropriate use of indexes and Hash clusters are important. Index-organized tables can also be important tuning options for large DSS systems. Bitmap indexes may be considered
where the column data has low cardinality but is frequently used as the basis for queries. Thought should also be given to selecting the appropriate optimizer mode and gathering new statistics whenever data is loaded or purged. The use of histograms may also help improve DSS response time. Finally, consideration should be given to database block size, init.ora parameters related to sorting, and the possible use of the Oracle Parallel Query option.

As with OLTP systems, database statistics should also be gathered following each data load if the CBO is used.

### Tuning Hybrid Systems: OLTP and DSS

Some systems are a combination of both OLTP and DSS. These hybrid systems can present a significant challenge because the tuning options that help one type of system frequently hinder the other. However, through the careful use of indexing and resource management, some level of coexistence can usually be achieved. As the demands on each of these systems grows, you will probably ultimately have to split the two types of systems into two separate environments in order to meet the needs of both user communities.

### Summary

Understanding how to use TKPROF, Explain Plans, STATSPACK, and AUTOTRACE to analyze application SQL processing is important for every DBA. These tools allow you to understand not only how the application SQL is being executed, but also what resource costs are associated with those executions.

One major determinant of database performance is the extent to which the Oracle optimizer makes good SQL execution choices. These choices are influenced by the optimizer mode and the presence or absence of database statistics. These statistics can be gathered at the segment, schema, or database level. The statistics can also be moved between databases so that testing of optimizer behavior can be done with realistic data.

A variety of methods are available to improve application performance. These methods include stored outlines, materialized views, B-Tree indexes, Bitmap indexes, function-based indexes, compressed indexes, Reverse Key indexes, index organized tables, Index clusters, and Hash clusters. The appropriateness of these options varies with the type of system managed. OLTP systems have different indexing strategies than DSS environments.
Exam Essentials

Describe the utilities for SQL Analysis. Understand how to use TKPROF, AUTORACE, Explain Plans, and STATSPACK to perform SQL analysis. Know how to interpret the output from each of these utilities in terms of potential performance tuning improvements.

Understand the role of the Optimizer. Know the importance of the Oracle optimizer and how to manage its behavior. Understand what optimizer modes a database can operate in and how each of these modes is impacted by the presence or absence of database statistics.

Know the different indexing strategies. Be able to describe each index type and how it can be used to improve SQL execution. Understand the situations in which each index type is appropriate.

Know the different tuning requirements. Understand how tuning goals are different between OLTP, DSS, and hybrid systems. Be able to place all the tuning considerations discussed in this chapter within the framework of each of these tuning environments.

Key Terms

Before you take the exam, be certain you are familiar with the following terms:

- Autotrace
- Bitmap
- blocks
- B-Tree
- Cardinality
- cluster
- composite partitioning
- Compressed
- cost-based optimizer (CBO)
- Decision Support Systems (DSS)
- Execute
- Explain Plan
- Fetch
- Function-based indexes
- Get
- Hash clusters
- Hash partitions
- Heap Tables
Hints                        partition pruning
Histogram                   partitions
Index clusters              plan equivalence
Index Organized Tables (IOTs) Private Outline
List partitions             Range partitioning
Mapping Table               Reverse Key Index (RKI)
Materialized views          rule-based optimizer
Online Transaction Processing stored outlines
Parsing                     TKPROF
Review Questions

1. Which of the following utilities allows you to see how much CPU time a SQL statement spent parsing, executing, and fetching?
   A. AUTOTRACE
   B. Explain Plan
   C. TKPROF
   D. Materialized view

2. In the statement tkprof ora_1234.trc trace.txt sys=no explain=joe/sidney@PROD, what does the SYS=NO do?
   A. Causes any recursive SQL in the original trace file to be omitted from the formatted trace file.
   B. Forces summary statistics not to be computed.
   C. Limits the number of SQL statements included in the resulting formatted trace.
   D. Generates an Explain Plan for each statement in the trace file.

3. Which TKPROF statistic provides the total number of seconds spent processing all the Parse, Execute, or Fetch calls in the statement?
   A. COUNT
   B. ELAPSED
   C. CURRENT
   D. ROWS

4. In order for the TKPROF output to include timing information, which init.ora parameter must be set to TRUE?
   A. TIMINGS
   B. TIME_QUERY
   C. STATISTICS
   D. TIMED_STATISTICS
5. An execution plan shows all blocks from a table being accessed in order to satisfy a query. What is this type of table access called?
   A. Full table scan
   B. Nested loop
   C. Range scan
   D. None of the above

6. What table should you query for information after running an Explain Plan on an SQL statement?
   A. SQL_PLAN
   B. SQL_PLAN_TABLE
   C. PLAN_TABLE
   D. EXECUTION_PLAN

7. What will be the result of executing the following SQL statement?
   SQL> ANALYZE TABLE EMP ESTIMATE STATISTICS
       2   SAMPLE 55 PERCENT;
   A. Oracle will estimate statistics for the EMP table and all indexes on the EMP table based on a 55 percent row sample.
   B. Oracle will estimate statistics for the EMP table only based on a 55 percent row sample.
   C. Oracle will compute statistics for the EMP table and all indexes on the EMP table for 100 percent of the rows.
   D. Oracle will compute statistics for the EMP table only for 100 percent of the rows.

8. Which of the following is not a valid value for the init.ora parameter OPTIMIZER_MODE?
   A. COST
   B. RULE
   C. CHOOSE
   D. All of the above are valid values.
9. If the `init.ora` optimizer mode is set to `CHOOSE`, but no statistics have been gathered for the application’s tables and indexes, which optimizer mode will Oracle use?
   A. COST
   B. RULE
   C. HINT
   D. None of the above

10. Which of the following Oracle packages can be used to analyze all the objects in a particular schema?
    A. `DBMS_PACKAGE`
    B. `DBMS_STATS`
    C. `DBMS_SYSTEM`
    D. `DBMS_ANALYZE`

11. Following an `ANALYZE` command, which of the following data dictionary views will contain statistics on the number of rows in a table?
    A. `DBA_ROWS`
    B. `DBA_TABLE_ROWS`
    C. `DBA_TABLES`
    D. `DBA_STATISTICS`

12. Histograms are useful because they allow the optimizer to have a better idea of whether the data in an indexed column is ____________ uniformly.
    A. sorted
    B. named
    C. entered
    D. distributed
13. The optimizer mode can be set at which of the following levels?
   A. Instance
   B. Session
   C. Statement
   D. All of the above

14. What is the ability to store predefined execution plans so that the optimizer knows in advance which execution plan is best called?
   A. Stored outlines
   B. Materialized views
   C. Hash cluster
   D. Index Organized Table

15. If you do not specify which category a stored outline belongs to, which category will it be assigned to?
   A. SYSTEM
   B. TEMP
   C. USERS
   D. DEFAULT

16. If a materialized view is completely refreshed each time a commit occurs in one of the tables that the view is based on, then this view was created with which option?
   A. REFRESH SYNC
   B. REFRESH FAST
   C. REFRESH COMPLETE
   D. REFRESH TOTAL

17. Which init.ora parameter must be set to TRUE if materialized views are to be used?
A. FAST_REFRESH
B. QUERY_REWRITE
C. QUERY_REWRITE_ENABLED
D. MATERIALIZED_VIEWS

18. Which of the following types of indexes stores the indexed column’s values and associated row IDs as a binary string?
   A. B-Tree index
   B. Reverse Key index
   C. Bitmap index
   D. Bitwise index

19. Which of the following factors would not make a column a good candidate for a B-Tree index?
   A. The data in the column has low cardinality.
   B. The column is frequently used in SQL statement WHERE clauses.
   C. Most queries on the table return only a small portion of all rows.
   D. None of the above.

20. Index-organized tables store row data that exceeds the index’s PCTTHRESHOLD in what type of segment?
   A. Chain Segment
   B. Excess Segment
   C. Overflow Segment
   D. Index segment

21. The Oracle9i Cost Based Optimizer uses all of the following statistics when determining an SQL statement’s cost except:
   A. Logical I/O
   B. Network load
   C. CPU Cost
   D. Amount of Sorting performed
22. The DBA wants to analyze a schema and collect statistics for only those tables that currently do not have statistics. What \texttt{DBMS\_STATS} procedure should be used to perform this activity?

\begin{itemize}
\item[A.] \texttt{GATHER\_SCHEMA\_STATS} with the \texttt{OPTIONS} parameter set to \texttt{GATHER EMPTY}
\item[B.] \texttt{GATHER\_SCHEMA\_STATS} with the \texttt{OPTIONS} parameter set to \texttt{GATHER AUTO}
\item[C.] \texttt{GATHER\_SYSTEM\_STATS} with the \texttt{OPTIONS} parameter set to \texttt{GATHER STALE}
\item[D.] \texttt{GATHER\_SYSTEM\_STATS} with the \texttt{OPTIONS} parameter set to \texttt{GATHER EMPTY}
\end{itemize}

23. What type of partitioning is being used in this example?

\texttt{SQL} CREATE TABLE student\_history
\begin{verbatim}
  (student\_id NUMBER(10),
   degree VARCHAR2(3),
   graduation\_date DATE,
   final\_gpa NUMBER)
PARTITION BY RANGE (graduation\_date)
PARTITION p\_1999 VALUES LESS THAN (TO\_DATE('01-JUN-1999','DD-MON-YYYY')) TABLESPACE hist\_tab01,
PARTITION p\_2000 VALUES LESS THAN (TO\_DATE('01-JUN-2000','DD-MON-YYYY')) TABLESPACE hist\_tab02,
PARTITION p\_2001 VALUES LESS THAN (TO\_DATE('01-JUN-2001','DD-MON-YYYY')) TABLESPACE hist\_tab03,
PARTITION p\_ERROR VALUES LESS THAN (MAXVALUE) TABLESPACE hist\_tab04);
\end{verbatim}

\begin{itemize}
\item[A.] Hash
\item[B.] Range
\item[C.] Composite
\item[D.] List
24. Given the partitioned table below, what would occur if a user attempted to insert a row with the value of MD into the degree column?

SQL> CREATE TABLE student_history
2 (student_id NUMBER(10),
3   degree VARCHAR2(3),
4   graduation_date DATE,
5   final_gpa NUMBER)
6   PARTITION BY LIST (degree)
7   (PARTITION p_undergrad VALUES ('BS','BA','BBA','BFA')
8     TABLESPACE hist_tab01,
9   PARTITION p_graduate VALUES ('MA','MBA','MFA','MS')
10   TABLESPACE hist_tab02,
11  PARTITION p_doctorate VALUES ('PHD')
12   TABLESPACE hist_tab03);

A. It would be placed in the last partition.

B. Oracle would automatically create a new partition with a partitioned value of MD.

C. Oracle would allow the insert of the value, but the COMMIT would fail.

D. Oracle would generate an error on the insert of the row.

25. Partitioned indexes are said to be ________________ when the number of partitions in the index match the underlying table’s partitions on a one-to-one basis.

A. Composite

B. Global

C. Local

D. Uniform
Answers to Review Questions

1. C. After formatting a user trace file using TKPROF, the formatted trace file will contain timings for both CPU and elapsed time for the Parse, Execute, and Fetch phases of a query (assuming the init.ora parameter TIMED_STATISTICS = TRUE).

2. A. The SYS=NO parameter for the TKPROF utility causes any recursive SQL in the original trace file to be omitted from the formatted trace file. An example of a recursive SQL statement would be calls made to the data dictionary during the execution of an SQL statement.

3. B. The ELAPSED TKPROF statistic is the total number of seconds spent processing all Parse, Execute, or Fetch calls in the statement.

4. D. If the init.ora parameter TIMED_STATISTICS is not set to TRUE, then all timings in a formatted TKPROF file will be 0.

5. A. Full table scans cause Oracle to read every row of a table. When full table scans are performed on a table, but very few of the total rows in the table are returned by the query, an appropriate indexing strategy should be investigated.

6. C. The PLAN_TABLE would be queried to provide information about execution plans generated using the EXPLAIN PLAN syntax. The PLAN_TABLE is created by running the utlxplan.sql script.

7. C. This statement would cause Oracle to compute statistics on the EMP table for the table and all of the indexes related to the table. The compute would be done because the sample percentage is greater than 50 percent.

8. A. The valid optimizer modes for the init.ora are RULE, CHOOSE, FIRST_ROWS, and ALL_ROWS.

9. B. Even if the optimizer mode is set to CHOOSE in the init.ora, the rule-based optimization will be used if statistics do not exist.

10. B. The DBMS_STATS package can be used to analyze all the objects in a given schema, index, or the entire database.
11. C. The DBA_TABLES data dictionary view also contains statistics about the number of blocks the table uses and the average length of each table row.

12. D. Histograms offer a better view of how data is distributed by dividing the column data into pieces before analyzing the data within each piece.

13. D. The optimizer mode can be set at the instance level using the init.ora parameter OPTIMIZER_MODE, at the session level using the ALTER SESSION command, or at the statement level using hints.

14. A. Stored outlines use the execution plan that was generated at their creation for any subsequent SQL statements that match the original SQL statement exactly.

15. D. Unless a category is assigned, stored outlines will be stored in the DEFAULT category. Stored outlines can be moved to another category using the OUTLN_PKG.UPDATE_BY_CAT procedure.

16. C. The REFRESH COMPLETE option causes a materialize view to be refreshed with each commit on the underlying tables. The REFRESH FAST option only updates the view with the data that has been added since the last refresh.

17. C. Materialized views can also be enabled at the statement level by using the /*+ REWRITE */ hint.

18. C. Bitmap indexes store their indexed column values as a binary string of 1s and 0s.

19. A. Columns with high cardinality are best suited for B-Tree indexing. Columns with low cardinality are better suited for Bitmap indexes.

20. C. Oracle will store any row whose length exceeds the value of PCTTHRESHOLD in an overflow segment. The first part of the row, up to the column specified in the INCLUDING clause, is stored in the index. The remainder of the row is stored in the overflow segment.

21. B. The Cost Based Optimizer (CBO) uses several weighting factors including Logical I/O, CPU Cost and the amount of sorting performed when creating SQL execution plans. It does not factor in network load for its calculation.
22. A. The DBA would use the GATHER_SCHEMA_STATS procedure with the OPTIONS parameter set to GATHER_EMPTY. This would only gather statistics for tables that did not have statistics already gathered.

23. B. This is an example of a Range partitioned table.

24. D. Oracle would not allow a value of MD to be placed in the table and would generate an ORA-14400 error upon the insert.

25. C. Local partitioned indexes have a partition key that matches the partitioning of the underlying table on a one-to-one basis.
Chapter 4

Tuning the Shared Pool

ORACLE9i PERFORMANCE TUNING EXAM OBJECTIVES COVERED IN THIS CHAPTER:

✓ Measure and tune the Library Cache hit ratio
✓ Measure and tune the Dictionary Cache hit ratio
✓ Size and pin objects in the Shared Pool
✓ Tune the Shared Pool reserve space
✓ Describe the UGA and session memory considerations
✓ Explain other tuning issues related to the Shared Pool

Exam objectives are subject to change at any time without prior notice and at Oracle’s sole discretion. Please visit Oracle’s Certification website (http://www.oracle.com/education/certification/) for the current exam objectives listing.
he previous chapter showed how to identify and remedy performance bottlenecks in an application’s SQL code. This is a very important part of database tuning, as rewriting poorly performing SQL can make a huge difference in how well an Oracle-based application performs. However, rewriting application SQL in an effort to improve database performance is frequently not an option, particularly when production applications are involved or when third-party applications, whose SQL code you may not have access to, are being tuned. In these cases, you have options when it comes to tuning:

- In the case of a purchased application, you can complain to the vendor and hope they do something about the poor performance.
- You can add your own indexes to the delivered product in the hopes that they will improve performance.
- You can tune Oracle’s shared memory structure, the SGA (System Global Area), and try to improve application performance by minimizing the overhead associated with SQL statement processing and the accessing of the application’s data and index blocks.

Realizing that the third option will probably result in the greatest performance gain, in the next four chapters we will address the issue of tuning the components of the Oracle SGA. This chapter deals with tuning the Shared Pool; Chapter 5, “Tuning the Database Buffer Cache,” discusses tuning the Database Buffer Cache, Chapter 6, “Tuning Other SGA Areas,” covers tuning the Large Pool, Java Pool, and Shared Server SGA components, and Chapter 7, “Tuning Redo Mechanisms,” discusses tuning the Redo Log Buffer.
Understanding the Shared Pool

The Shared Pool is the portion of the SGA that caches the SQL and PL/SQL statements that have been recently issued by application users. The Shared Pool is managed by a Least Recently Used (LRU) algorithm. Once the Shared Pool is full, this algorithm makes room for subsequent SQL statements by aging out the statements that have been least recently accessed. In this manner, the Oracle server keeps the “popular” SQL and PL/SQL statements in the Shared Pool while the less commonly used statements are phased out. By caching these frequently used statements in memory, an application user who issues the same SQL or PL/SQL statement as that of a previous user benefits from the fact that the statement is already in memory, ready to be executed. This is very useful for the subsequent application user as that user can skip some of the overhead that was incurred when the statement was originally issued.

The Benefits of Statement Caching

When an application user issues an SQL or PL/SQL statement against an Oracle instance, several operations take place before the statement is actually executed. For example, let’s examine what happens when the following SQL statement is issued:

```
SQL> SELECT last_name, first_name
       2 FROM customers
       3 WHERE customer_id = 2201;
```

First, Oracle converts the characters in the statement to their ASCII equivalent numeric codes. Next, this string of ASCII characters is passed through a hashing algorithm, which in turn produces a single hashed value. The user’s Server Process then checks to see if that hashed value already exists in the Shared Pool. If it does, the user’s Server Process uses the cached version of the statement to execute the statement.

If the hashed value does not exist in the Shared Pool, the user’s Server Process goes about the process of parsing the statement, then executes it. This extra step of parsing adds overhead to each SQL operation because the parse step must complete the following tasks:

- Check the statement for syntactic correctness.
Perform object resolution where the names and structures of the referenced objects are checked against the data dictionary.

Gather statistics regarding the objects referenced in the query by examining the data dictionary.

Prepare and select an execution plan from among the available plans, including the determination of whether Stored Outlines or Materialized Views are relevant.

Determine the security for the objects referenced in the query by examining the data dictionary.

Generate a compiled version of the statement (called P-Code).

The parse operation is expensive because of this extra preparation. Parsing can be avoided by making sure that most statements find a parsed version of themselves already in the Shared Pool. Finding a matching SQL statement in the Shared Pool is referred to as a cache hit. Not finding a matching statement and having to perform the parse operation is considered a cache miss. In order for a cache hit to occur, the two SQL statements must match exactly; the ASCII equivalents of the two statements must be identical for them to hash to the same value in the Shared Pool. Therefore, even a minor difference between two statements, like lowercase versus uppercase, will cause a cache miss to occur. Maximizing cache hits and minimizing cache misses is the goal of all Shared Pool tuning.

As of Oracle9i, a hashed value is also assigned to the execution plan for each cached SQL statement. As with SQL statement hash values, Oracle can use this hashed value to determine whether two execution plans are identical.

However, before you can begin to tune the Shared Pool, you must first understand what functions each component of the Shared Pool performs.

**Components of the Shared Pool**

The Shared Pool is made up of three components: the Library Cache, the Data Dictionary Cache, and the User Global Area. Each of these components plays a role in the performance of SQL and PL/SQL statements.
Library Cache

The Shared Pool’s Library Cache is the location where Oracle caches the SQL and PL/SQL statements that have been recently issued by application users. PL/SQL statements can be in the form of procedures, functions, packages, triggers, anonymous PL/SQL blocks, or Java classes. Once cached, these cached statements have several components:

- The actual text of the statement itself
- The hashed value associated with the statement
- The P-Code for the statement
- Any statistics associated with the statement
- The execution plan for the statement

Many of these components for an SQL statement can be viewed in four dynamic performance views: V$SQL, V$SQLAREA, V$SQL_TEXT, and V$SQL_PLAN. Table 4.1 compares the contents of these four views, all of which contain information about the SQL statements cached in the Library Cache.

**TABLE 4.1** Contents of the SQL-Related Dynamic Performance Views

<table>
<thead>
<tr>
<th>View Name</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>V$SQL</td>
<td>SQL statement text and statistics for all cached SQL statements including I/O, memory usage, and execution frequency</td>
</tr>
<tr>
<td>V$SQLAREA</td>
<td>Similar to V$SQL, but displays only information for SQL statements that are cached in memory, parsed, and ready for execution</td>
</tr>
<tr>
<td>V$SQLTEXT</td>
<td>The complete text of the cached SQL statements along with a classification by command type</td>
</tr>
<tr>
<td>V$SQL_PLAN</td>
<td>Execution plan information for each cached SQL statement</td>
</tr>
</tbody>
</table>
These views can be used to see which statements application users are actually issuing against the instance, how many times a statement has been issued, and how many disk and buffer reads and sorts a statement did.

The V$SQLAREA view contains only the first 80 characters of the SQL or PL/SQL statements that are being issued. The complete text of these SQL or PL/SQL statements can be found in the SQL_TEXT column of the data dictionary view V$SQLTEXT.

The execution plans that are cached in the Shared Pool can be seen by querying the V$SQL_PLAN dynamic performance view. This view contains most of the columns found in an EXPLAIN_PLAN table, plus some additional columns used to keep track of which cached SQL statement each execution plan belongs to. These views can even be joined together in order to get a more complete picture of the SQL activity against the database. For example, the query shown in Figure 4.1 displays the username, SQL text, and execution plan of a SQL statement cached in the Library Cache by the user HR.

**FIGURE 4.1** Examining the contents of the Library Cache

```sql
SQL> SELECT p.username, LPAD(' ',4*(LEVEL-2))||
          operation||' '||
          options||' '||
          object_name "Execution Plan"
FROM {
    SELECT s.username, p.address, p.hash_value, p.operation,
          p.options, p.object_name, p.id, p.pp
    FROM v$sql_plan p, v$session s
    WHERE (p.address = s.sql_address
          AND p.hash_value = s.sql_hash_value)
          AND s.username = 'HR') p, v$sql
WHERE (p.address = p.address
      AND (p.hash_value = p.hash_value)
      START WITH id = 0,
      CONNECT BY PRIOR id = parent_id;
```

If you are only looking for more general information about the types of SQL statements application users are issuing, you can use the V$SESSION and V$DB_OBJECT_CACHE views.
The V$SESSION view contains general information about each of the user processes that are currently connected to the instance. The V$SESSION column, COMMAND, can be used to identify generically what types of statements users are issuing. There are approximately 90 possible values for COMMAND; each represents a different type of SQL command. Table 4.2 shows a sample of some of the values that you may want to use when querying the COMMAND column in this view.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Insert</td>
</tr>
<tr>
<td>3</td>
<td>Select</td>
</tr>
<tr>
<td>6</td>
<td>Update</td>
</tr>
<tr>
<td>7</td>
<td>Delete</td>
</tr>
<tr>
<td>26</td>
<td>Lock table</td>
</tr>
<tr>
<td>44</td>
<td>Commit</td>
</tr>
<tr>
<td>45</td>
<td>Rollback</td>
</tr>
<tr>
<td>47</td>
<td>PL/SQL Execute</td>
</tr>
</tbody>
</table>

For example, you could use the following query on the V$SESSION view to see which users are currently issuing UPDATE commands:

```
SQL> SELECT username
2  FROM v$session
3  WHERE command = 6;

USERNAME
--------
ZOE
MAX
ROBIN
JIM
```
The V$DB_OBJECT_CACHE view can be used to determine what types of database objects are being referenced by the application’s SQL statements. By selecting the TYPE and EXECUTIONS columns from this view, you can obtain a listing of how often the application SQL has referenced each of the database object types:

```
SQL> SELECT type, COUNT(executions) FROM v$db_object_cache GROUP BY type ORDER BY 2 DESC
```

<table>
<thead>
<tr>
<th>TYPE</th>
<th>COUNT(EXECUTIONS)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CURSOR</td>
<td>271</td>
</tr>
<tr>
<td>NOT LOADED</td>
<td>91</td>
</tr>
<tr>
<td>TABLE</td>
<td>73</td>
</tr>
<tr>
<td>INDEX</td>
<td>31</td>
</tr>
<tr>
<td>VIEW</td>
<td>29</td>
</tr>
<tr>
<td>SYNONYM</td>
<td>13</td>
</tr>
<tr>
<td>CLUSTER</td>
<td>6</td>
</tr>
<tr>
<td>PACKAGE</td>
<td>4</td>
</tr>
<tr>
<td>PUB_SUB</td>
<td>4</td>
</tr>
<tr>
<td>NON-EXISTENT</td>
<td>2</td>
</tr>
<tr>
<td>PACKAGE BODY</td>
<td>2</td>
</tr>
<tr>
<td>SEQUENCE</td>
<td>2</td>
</tr>
<tr>
<td>PROCEDURE</td>
<td>1</td>
</tr>
<tr>
<td>TRIGGER</td>
<td>1</td>
</tr>
</tbody>
</table>

The high number of CURSOR executions in conjunction with the low number of PACKAGE_BODY and PACKAGE executions indicates most of the application’s SQL is issued via straight SQL statements, not through the use of stored procedures or functions. This information is useful because converting some of this straight SQL into PL/SQL packages or functions may help improve Shared Pool performance. This concept is discussed in the “Improving Shared Pool Performance” section of this chapter.

**Note:** Detailed information and the dynamic performance views V$SQL, V$SQLAREA, V$SQLTEXT, V$DB_OBJECT_CACHE, V$SQL_PLAN, and V$SESSION, can be found in Chapter 3, “Dynamic Performance (V$) Views,” of the Oracle9i Database Reference, Release 1 (9.0.1) (Part Number A90190-01).
Data Dictionary Cache

Whenever an SQL or PL/SQL statement is issued, the data dictionary must also be examined to make sure that the tables referenced in the statement exist, that the column names and data types are correct, and that the application user issuing the statement has sufficient privileges on the segments involved. Like the statement itself, this information is also cached in memory in the Shared Pool’s Data Dictionary Cache. This memory area is also managed using an LRU mechanism.

By using the Data Dictionary Cache, the Oracle Server caches the data dictionary information in a separate memory area than it does the associated SQL statements. This is useful for two reasons:

- The LRU mechanisms for the Library Cache and Data Dictionary Cache work independently. This tends to keep data dictionary information in memory longer.
- Subsequent application users who issue similar, but not identical, statements as previous users benefit from the fact that the data dictionary information associated with the tables in their statement may be in memory—even if the actual statement is not.

User Global Area

The User Global Area (UGA) is only present in the Shared Pool if the Shared Server option is being used. In the shared server architecture, the User Global Area is used to cache application user session information. This information must be in a shared location because the shared server architecture uses several different shared server processes to process SQL and PL/SQL activity against the instance. Since the shared server process that starts a transaction may not be the one that also finishes it, access to this session information must be stored in the UGA. In a non-shared (or dedicated) server situation, this session information is maintained in the application user’s private Process Global Area (PGA).

See Chapter 6 of this book, “Tuning Other SGA Areas,” for more details on tuning a shared server configuration.
Measuring the Performance of the Shared Pool

The primary indicator of the performance of the Shared Pool is the cache-hit ratio. High cache-hit ratios indicate that your application users are frequently finding the SQL and PL/SQL statements they are issuing already in memory. Hit ratios can be calculated for both the Library Cache and the Data Dictionary Cache.

Oracle recommends tuning the Library Cache to maximize its hit ratio before attempting to tune the Database Buffer Cache because a less-than-optimal cache hit ratio for the Library Cache has a greater impact on performance than does an underperforming cache hit ratio for the Database Buffer Cache.

Measuring Library Cache Performance

The performance of the Library Cache is measured by calculating its hit ratio. This hit ratio information can be found in four places: the V$LIBRARYCACHE dynamic performance view, STATSPACK and REPORT.TXT output, and the output from the Performance Manager GUI tool.

Oracle Objective

Measure and tune the Library Cache hit ratio

Using V$LIBRARYCACHE to Monitor the Library Cache

A Library Cache miss can occur at either the parse or execute phases of SQL statement processing. The cache hit ratio related to the parse phase is shown in the GETHITRATIO column of V$LIBRARYCACHE. The cache hit ratio related to the execution phase is shown in the PINHITRATIO column of V$LIBRARYCACHE. The RELOADS and INVALIDATIONS columns of V$LIBRARYCACHE are also useful for Library Cache tuning. A sample query on V$LIBRARYCACHE showing these columns is included here in Figure 4.2.
Measuring the Performance of the Shared Pool

**FIGURE 4.2** Sample query on V$LIBRARYCACHE

```sql
SQL> SELECT namespace, gethitratio, pinhitratio, reloads, invalidations
  2  FROM v$librarycache
  3  WHERE namespace IN ('SQL AREA',
  4    'TABLE/PROCEDURE',
  5    'BODY',
  6    'TRIGGER');
```

<table>
<thead>
<tr>
<th>Namespace</th>
<th>GETHITRATIO</th>
<th>PINHITRATIO</th>
<th>RELOADS</th>
<th>INVALIDATIONS</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQL AREA</td>
<td>0.915877487</td>
<td>0.097509722</td>
<td>10</td>
<td>2</td>
</tr>
<tr>
<td>TABLE/PROCEDURE</td>
<td>0.799803656</td>
<td>0.087164434</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>BODY</td>
<td>0.7</td>
<td>0.025</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>TRIGGER</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

The NAMESPACE column in the above figure shows the source of the cached code. Table 4.3 shows the definition of each namespace type.

**TABLE 4.3** Code Source by Namespace Type

<table>
<thead>
<tr>
<th>Namespace</th>
<th>Code Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQL AREA</td>
<td>SQL statement</td>
</tr>
<tr>
<td>TABLE/PROCEDURE</td>
<td>PL/SQL stored procedure or function</td>
</tr>
<tr>
<td>BODY</td>
<td>PL/SQL package body</td>
</tr>
<tr>
<td>TRIGGER</td>
<td>PL/SQL trigger</td>
</tr>
</tbody>
</table>

The other values for NAMESPACE found in V$LIBRARYCACHE refer to information used to determine dependencies between cached objects and statements. These values are of little value to you in terms of database tuning.

**GETHITRATIO**

In V$LIBRARYCACHE, Oracle uses the term get to refer to a type of lock, called a *Parse Lock*, that is taken out on an object during the parse phase for the statement that references that object. Each time a statement is parsed, the value for GETS in the V$LIBRARYCACHE view is incremented by 1. The column GETHIT stores the number of times that the SQL and PL/SQL statements issued by the application found a parsed copy of themselves already in
memory. When this occurs, there is no parsing of the statement required. The user’s Server Process just executes the copy already in memory instead. The ratio of parsed statements (GETS) to those that did not require parsing (GETHITS) is calculated in the GETHITRATIO column of V$LIBRARYCACHE. In the sample output in Figure 4.2, our application users are finding the SQL statements they are issuing already in the Library Cache (i.e., GETHITRATIO) 91.5 percent of the time. The higher this number is, the better the application is performing.

According to Oracle, well-tuned OLTP systems can expect to have GETHITRATIOS of 90 percent or higher for the SQL Area portion of the Library Cache. DSS and data warehouse applications often have lower hit ratios because of the ad hoc nature of the queries against them.

**PINHITRATIO**

PINS, like GETS, are also related to locking. However, while GETS are associated with locks that occur at parse time, PINS are related to locks that occur at execution time. These locks are the short-term locks used when accessing an object. Therefore, each library cache GET also requires an associated PIN, in either Shared or Exclusive mode, before accessing the statement’s referenced objects. Each time a statement is executed, the value for PINS is incremented by 1. The PINHITRATIO column in V$LIBRARYCACHE indicates how frequently executed statements found the associated parsed SQL already in the Library Cache. In the sample output in Figure 4.2, this is occurring 93.7 percent (i.e., PINHITRATIO) of the time.

Oracle recommends that well-tuned OLTP systems strive for a PINHITRATIO that exceeds 90 percent for the SQL Area portion of the Library Cache. DSS and data warehouse applications will generally be lower.

**RELOADS**

The Reloads column shows the number of times that an executed statement had to be re-parsed because the Library Cache had aged out or invalidated the parsed version of the statement. Reload activity can be monitored by
comparing the number of statements that have been executed (PINS) to the number of those statements that required a reload (RELOADS):

```sql
SQL> SELECT SUM(reloads)/SUM(pins) "Reload Ratio"
      2 FROM v$librarycache;
```

Reload Ratio
------------
  .001506251

In this example our Reload Ratio is .0015, or 0.15 percent. This means that we are not re-parsing statements that were previously loaded in the Library Cache very often.

Oracle considers well-tuned systems to be those with Reload Ratios of less than 1 percent.

**INVALIDATIONS**

Invalidations occur when a cached SQL statement is marked as invalid and therefore forced to parse even though it was already in the Library Cache. Cached statements are marked as invalid whenever the objects they reference are modified in some way. For example, recompiling a view that was used by previously cached SQL statements will cause those cached statements to be marked as invalid. Therefore, any subsequent statements that use this view will need to be parsed even though an exact copy of that statement may already be cached. High values for INVALIDATIONS mean additional overhead for the application. Therefore, performing activities that might cause invalidations should be weighed against the expected benefit of those activities.

**Using STATSPACK to Monitor the Library Cache**

Much of the information obtained from the preceding queries against V$LIBRARYCACHE is also available in the output from the STATSPACK utility. Library Cache performance statistics are included in the STATSPACK section headed *Instance Efficiency Percentages (Target 100%)*. The STATSPACK output shown in Figure 4.3 shows that the Library Cache hit ratio was 86.11 percent for the database being monitored.
FIGURE 4.3  STATSPACK output showing Library Cache hit ratio

<table>
<thead>
<tr>
<th>Instance Efficiency Percentages (Target 100%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Buffer Nowait %: 100.00</td>
</tr>
<tr>
<td>Buffer Hit %: 100.00</td>
</tr>
<tr>
<td>Library Hit %: 86.11</td>
</tr>
<tr>
<td>Execute To Parse %: 42.96</td>
</tr>
<tr>
<td>Parse CPU to Parse Elapsed %:</td>
</tr>
</tbody>
</table>

```
<table>
<thead>
<tr>
<th>shared pool statistics</th>
<th>begin</th>
<th>end</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory Usage %:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>% SQL with executions:</td>
<td>53.78</td>
<td>58.42</td>
</tr>
<tr>
<td>% Memory for SQL w/execs:</td>
<td>63.47</td>
<td>75.74</td>
</tr>
</tbody>
</table>
```

STATSPACK also includes a second section, headed by Library Cache Activity for DB, which shows the same information about the invalidations and reloads that appeared in the V$LIBRARYCACHE queries shown in the previous section. Figure 4.4 shows sample STATSPACK output from a database called PROD.

FIGURE 4.4  STATSPACK output showing Library Cache invalidations and reloads

```
GetReq Misses: should be very low

<table>
<thead>
<tr>
<th>Namespace</th>
<th>GET Requests</th>
<th>GET Miss</th>
<th>PIN Requests</th>
<th>PIN Miss</th>
<th>RELOADS</th>
<th>INVALIDATIONS</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQL AREA</td>
<td>75</td>
<td>2.8</td>
<td>438</td>
<td>11.4</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>TABLE/PROCEDURE</td>
<td>219</td>
<td>0.5</td>
<td>390</td>
<td>16.9</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
```

This section of STATSPACK also shows information about the Get ratio for the Library Cache. However, in the STATSPACK output the performance is indicated in terms of a miss ratio instead of a hit ratio as previously described. Figure 4.4 shows that the value for Pct Miss for the Get Requests activity of the SQL AREA namespace is 2.8 or 2.8 percent. Conversely, this means that the SQL AREA portion of the Library Cache had a 97.2 percent hit ratio during the monitoring period—well within the 90 percent guideline recommended by Oracle.

Using REPORT.TXT to Monitor the Library Cache

The REPORT.TXT file generated by running UTLBSTAT.SQL and UTLESTAT.SQL also contains a section that shows GETHITRATIO, PINHITRATIO, RELOADS, and INVALIDATIONS information for the Library Cache. Figure 4.5 shows the relevant section of a REPORT.TXT file.
Measuring the Performance of the Shared Pool

**FIGURE 4.5** *REPORT.TXT* output showing Library Cache statistics

```sql
SQL> column library format a12 trunc;
SQL> column pinnratio heading 'PNHITRAT';
SQL> column getsratio heading 'GETHRAT';
SQL> column invalidations heading 'INVALIDAT';
SQL> set numwidth 10;
SQL> from select library,  
gets,  
round(decode(gethits,0,1,gethits)/decode(gets,0,1,gets),3) getsratio,  
pins,  
round(decode(pfnhits,0,1,pfnhits)/decode(pins,0,1,pins),3) pinnratio,  
reloads, invalidations  
from stats$1lib;

<table>
<thead>
<tr>
<th>library</th>
<th>GETS</th>
<th>GETHRAT</th>
<th>PINS</th>
<th>PNHITRAT</th>
<th>RELOADS</th>
<th>INVALIDAT</th>
</tr>
</thead>
<tbody>
<tr>
<td>BODY</td>
<td>10</td>
<td>1</td>
<td>10</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>CLUSTER</td>
<td>176</td>
<td>.989</td>
<td>258</td>
<td>.984</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>INDEX</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>JAVA DATA</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>JAVA RESOURC</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>JAVA SOURCE</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>OBJECT</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>PIPE</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SQL AREA</td>
<td>3669</td>
<td>.939</td>
<td>88350</td>
<td>.933</td>
<td>36</td>
<td>0</td>
</tr>
<tr>
<td>TABLE/PROCED</td>
<td>1224</td>
<td>.859</td>
<td>7663</td>
<td>.954</td>
<td>43</td>
<td>0</td>
</tr>
<tr>
<td>TRIGGER</td>
<td>82</td>
<td>.89</td>
<td>82</td>
<td>.829</td>
<td>5</td>
<td>0</td>
</tr>
</tbody>
</table>
```

This information is essentially identical to the information contained in the `V$LIBRARYCACHE` view. However, unlike a query on `V$LIBRARYCACHE`, which shows the Library Cache statistics since instance startup, the Library Cache statistics contained in `REPORT.TXT` are only for the period between the times when `UTLBSTAT.SQL` and `UTLESTAT.SQL` were executed.

**Using Enterprise Manager to Monitor the Library Cache**

The Performance Manager component of the Oracle Enterprise Manager GUI tool contains several sources of Library Cache performance information. The first, shown in Figure 4.6, is the output from the SGA Overview option located in the Memory category of the Performance Manager utilities.

The output shown in Figure 4.6 indicates that the Shared Pool % Free value displayed in the upper-right panel is close to 100 percent. This indicates that the Shared Pool has ample room remaining to cache additional application SQL statements.

Another useful component of the Performance Manager, Library Cache Hit %, is found in the Database Instance category of the Performance Manager utilities. Figure 4.7 shows sample output from this utility.
**Figure 4.6** SGA Overview from OEM Performance Manager

![SGA Overview](image1)

**Figure 4.7** Library Cache Hit % from OEM Performance Manager

![Library Cache Hit](image2)
Figure 4.7 shows a graph of the Library Cache hit ratio after it was monitored over a particular interval of time. This graph shows that the Library Cache hit ratio hovered around 100 percent for most of the monitoring period, but dipped to approximately 90 percent briefly near the beginning of the monitoring period.

Tools in Performance Manager’s SQL category are also useful for monitoring Library Cache activity. In particular, the SQL Hard Parse Analysis option of the SQL category shows how often parsing was occurring for application SQL statements. Figure 4.8 shows sample output of the SQL Hard Parse Analysis option.

FIGURE 4.8 SQL Hard Parse Analysis from OEM Performance Manager

The output in Figure 4.8 displays a breakdown of the current SGA size along with SQL fragments of the hard-parsed code along with the number of hard parses for each database session. If a particular user session shows evidence of high hard-parse counts, then the application code being executed by that user should be examined for possible enhancements that will encourage SQL reuse instead of hard parses. Techniques for encouraging SQL code reuse are described later in this chapter.
Measuring Data Dictionary Cache Performance

Like the Library Cache, the measure of the effectiveness of the Data Dictionary Cache is expressed in terms of a hit ratio. This Dictionary Cache hit ratio shows how frequently the application finds the data dictionary information it needs in memory, instead of having to read it from disk. This hit-ratio information is contained in a dynamic performance view called V$ROWCACHE. The output from STATSPACK and REPORT.TXT also contain statistics related to the Data Dictionary Cache hit ratio.

| Oracle Objective | Measure and tune the Dictionary Cache hit ratio |

Using V$ROWCACHE to Monitor the Data Dictionary Cache

Two of the columns contained in the V$ROWCACHE view, GETS and GETMISSES, can be used to calculate the Data Dictionary Cache hit ratio. The statistics in this view are not generally examined individually, but in aggregate using the query shown here:

```sql
SQL> SELECT 1 - (SUM(getmisses)/SUM(gets))
       2  "Data Dictionary Hit Ratio"
       3  FROM v$rowcache;
```

Data Dictionary Hit Ratio
--------------------------
.969700203

The result of this query shows that your application is finding the data dictionary information it needs, already in memory, 96.9 percent of the time.

Oracle recommends that consideration be given to tuning the Shared Pool if the overall Data Dictionary Cache hit ratio is less than 85 percent.

Using STATSPACK to Monitor the Data Dictionary Cache

Data Dictionary Cache performance indicators can also be found in the output from the STATSPACK utility in the section headed Dictionary Cache.
Stats for DB. Sample output displaying this section of a STATSPACK report is shown in Figure 4.9.

**FIGURE 4.9**  STATSPACK output showing Data Dictionary Cache hit ratio

Like the output for the Library Cache, the STATSPACK output for Data Dictionary Cache performance is also expressed in terms of a miss ratio. Figure 4.9 shows that the value for \textit{Pct Miss} for each of the components of the Data Dictionary Cache (e.g., \texttt{dc\_free\_extents}, \texttt{dc\_object\_ids}) ranges from a zero to 3.6 percent miss ratio. Conversely, this means that the Data Dictionary Cache hit ratio was between 96.4 and 100 percent during the monitoring period.

Oracle recommends that consideration be given to tuning the Shared Pool if the STATSPACK output shows that the hit ratio for any single component of the Data Dictionary Cache (e.g., \texttt{dc\_free\_extents}, \texttt{dc\_object\_ids}) is lower than 98 percent for most components.

**Using \textit{REPORT.TXT} to Monitor the Data Dictionary Cache**

Data Dictionary information is also contained in the \textit{REPORT.TXT} output produced by \texttt{UTLSTAT.SQL} and \texttt{UTLESTAT.SQL}. Like STATSPACK, \textit{REPORT.TXT} displays the Data Dictionary Cache statistics separately for each component that is cached. Therefore, you will need to add the \texttt{GET\_REQS} and \texttt{GET\_MISS} columns yourself before calculating the overall hit ratio. Figure 4.10 shows the relevant section of a \textit{REPORT.TXT} file.
Using this output and the formula \(1 - \frac{\text{Sum of the GET\_MISS column}}{\text{Sum of the GET\_REQS column}}\), you can calculate the Data Dictionary Cache hit ratio of this database to be 96.9 percent.

### Improving Shared Pool Performance

Now that you know about the components of the Shared Pool and can measure the performance of each, you can examine methods for improving Shared Pool performance. The objective of all these methods is to increase Shared Pool performance by improving Library Cache and Data Dictionary Cache hit ratios. These techniques fall into five categories:

- Make it bigger
- Make room for large PL/SQL statements
- Keep important PL/SQL code in memory
- Encourage code reuse
- Tune Library Cache–specific `init.ora` parameters
Improving Shared Pool Performance

Make It Bigger

The simplest way to improve the performance of the Library and Data Dictionary Caches is to increase the size of the Shared Pool. Because the Oracle Server dynamically manages the relative sizes of the Library and Data Dictionary Caches from amongst the total space available to the Shared Pool, making the Shared Pool larger lessens the likelihood that cached information will be moved out of either cache by the LRU mechanism. This has the effect of improving both the Library Cache and Data Dictionary Cache hit ratios.

It is rare to see a situation where the Library Cache’s hit ratios are good, but the Data Dictionary Cache’s are bad, or vice versa. The hit ratios for both of these caches tend to move together as a result of this dynamic allocation of space between the two caches.

The size of the Shared Pool is determined by the *init.ora* parameter `SHARED_POOL_SIZE`. This parameter is specified in bytes and by default is set to 64MB on 64-bit operating systems and 16MB on 32-bit operating systems. The current size of the Shared Pool can be determined by querying the `V$SGASTAT` dynamic performance view:

```
SQL> SELECT pool, sum(bytes) 'SIZE'
    2  FROM v$sgastat
    3  WHERE pool = 'shared pool'
    4  GROUP BY pool;

POOL             SIZE
---------------- ----------
shared pool  209715200
```

If you are using Oracle’s Java capabilities, it is important to make sure the Shared Pool is at least 50MB in size. Over 4,000 Java classes are loaded at instance startup, and they require at least this much Shared Pool space to function properly. By default, Oracle9i configures a third pool, the Java Pool, of size 20,000 bytes. The size of the Java Pool is specified by the *init.ora* parameter `JAVA_POOL_SIZE`. This pool is used to cache Java state and runtime execution information.
See Chapter 6, "Oracle9i Java Application Performance," in Oracle9i Java Developers Guide, Release 1 (Part Number A90209-01) for more information on sizing the Java Pool.

The V$DB_OBJECT_CACHE and V$SQLAREA dynamic performance views discussed in the previous section can be used to arrive at an estimated size for the Library Cache. Figure 4.11 shows an anonymous PL/SQL block that you can use to determine how much sharable memory the instance’s cached SQL statements are currently utilizing:

**FIGURE 4.11** Using V$ Views to estimate proper Shared Pool size

```sql
set echo off
set feedback off
set serveroutput on

DECLARE
  v_total_plsql_mem NUMBER := 0;
  v_total_sql_mem NUMBER := 0;
  v_total_sharable_mem NUMBER := 0;
BEGIN
  -- Find the total sharable memory used by non-SQL objects like
  -- packages, views, etc.
  SELECT sum(sharable_mem)
  INTO v_total_plsql_mem
  FROM v$object_cache;

  -- Find the total sharable memory used by SQL statements
  SELECT sum(sharable_mem)
  INTO v_total_sql_mem
  FROM v$sqlarea
  WHERE executions > 10;

  -- Now add these two values to get the total sharable memory
  -- (i.e., Library Cache) requirements
  v_total_sharable_mem := v_total_plsql_mem + v_total_sql_mem;

  DBMS_OUTPUT.PUT_LINE
  ('Estimated required Shared Pool size is: ' || TO_CHAR(v_total_sharable_mem, '999,999,999,999') || ' bytes');
END;
```

The above query totals all the sharable memory used by non-SQL objects and the sharable memory used by any SQL statement that has been executed more than ten times. This helps eliminate from consideration any extraneous, infrequently executed SQL statements that may not be part of the core application code. Figure 4.12 shows sample output from the execution of this anonymous PL/SQL block.
In addition to the value returned by the previous query, Oracle recommends that you add an additional 250 bytes to the Shared Pool size estimate for every expected concurrent user session.

Using these results, the size of the Shared Pool can be adjusted either dynamically or manually via an instance shutdown and startup.

**Dynamically Increasing the Shared Pool Size**

The size of the Shared Pool can be changed dynamically using the ALTER SYSTEM command. The following example would change the Shared Pool size to 200MB:

```
SQL> ALTER SYSTEM SET shared_pool_size = 200M;
```
When adjusting the size of the Shared Pool, the total size of the new Shared Pool, Database Buffer Cache and Redo Log Buffer cannot exceed the value specified by SGA_MAX_SIZE.

Attempting to increase the size the Shared Pool past the limit imposed by the SGA_MAX_SIZE parameter in the init.ora will cause an ORA-04033: Insufficient memory to grow pool error.

**Manually Increasing the Shared Pool Size**

The Shared Pool can also be resized by shutting down the instance, changing the value for SHARED_POOL_SIZE, and then restarting the instance. This technique is required whenever you need to increase the value of SGA_MAX_SIZE so that a larger value for SHARED_POOL_SIZE can be specified.

Shutting down and then restarting the instance will produce artificially low Shared Pool hit ratios until the instance “warms up” and the Shared Pool has reloaded application SQL statements.

---

**Real World Scenario**

**How Big Should I Make the Shared Pool on My New Server?**

When setting up a new server where no previous statistics are available, you are forced to take educated guesses at the sizing of SGA components like the Shared Pool. While Oracle provides some suggested values (small, medium, and large) for the various SGA components in its sample init.ora, these are rarely useful since “small, medium, and large” are only relative terms.

**Initial SGA Size Calculation**

Instead of using the Oracle sample values, I use the following rule of thumb when setting the SGA size for a new system:

- Server Physical Memory $\times .55 = \text{Total Amount of Memory to Allocate to All SGAs (TSGA)}$
- $\frac{\text{TSGA}}{\text{Number of Oracle Instances on the Server}} = \text{Total SGA Size Per Instance (TSGAI)}$
Improving Shared Pool Performance

Make Room for Large PL/SQL Statements

If an application makes a call to a large PL/SQL package or trigger, several other cached SQL statements may be moved out of memory by the LRU mechanism when this large package is loaded into the Library Cache. This has the effect of reducing the Library Cache hit ratio if these statements are subsequently re-read into memory later. To avoid this problem, Oracle gives you the ability to set aside a portion of the Library Cache for use by large PL/SQL packages. This area of the Shared Pool is called the *Shared Pool Reserved Area*.

Oracle Objective

| Tune the Shared Pool reserve space |
The init.ora parameter `SHARED_POOL_RESERVED_SIZE` can be used to set aside a portion of the Shared Pool for exclusive use by large PL/SQL packages and triggers. The value of `SHARED_POOL_RESERVED_SIZE` is specified in bytes and can be set to any value up to 50 percent of the value specified by `SHARED_POOL_SIZE`. By default, `SHARED_POOL_RESERVED_SIZE` will be 5 percent of the size of the Shared Pool.

As a starting point, Oracle recommends setting `SHARED_POOL_RESERVED_SIZE` to 10 percent of the `SHARED_POOL_SIZE`.

One way to determine the optimal size of your reserve pool is to monitor the `V$DB_OBJECT_CACHE` dynamic performance view. The following query can be used to see the names and sizes of the PL/SQL packages that are currently cached in memory:

```
SQL> SELECT owner, name, sharable_mem
2  FROM v$db_object_cache
3  WHERE type IN ('PACKAGE', 'PACKAGE BODY')
4  ORDER BY SHARABLE_MEM DESC;
```

<table>
<thead>
<tr>
<th>OWNER</th>
<th>NAME</th>
<th>SHARABLE_MEM</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYS</td>
<td>STANDARD</td>
<td>218260</td>
</tr>
<tr>
<td>SYS</td>
<td>DBMS_SPACE_ADMIN</td>
<td>19312</td>
</tr>
<tr>
<td>SYS</td>
<td>DBMS_STANDARD</td>
<td>15809</td>
</tr>
<tr>
<td>SYS</td>
<td>DBMS_OUTPUT</td>
<td>14171</td>
</tr>
<tr>
<td>SYS</td>
<td>DBMS_APPLICATION_INFO</td>
<td>12629</td>
</tr>
<tr>
<td>SYS</td>
<td>DBMS_OUTPUT</td>
<td>6231</td>
</tr>
<tr>
<td>SYS</td>
<td>DBMS_SPACE_ADMIN</td>
<td>5324</td>
</tr>
<tr>
<td>SYS</td>
<td>DBMS_APPLICATION_INFO</td>
<td>2865</td>
</tr>
</tbody>
</table>

The dynamic performance view `V$SHARED_POOL_RESERVED` can be used to monitor the use of the Shared Pool Reserved Area and help determine if it is properly sized. Indications that you have over-allocated memory for the reserved area include:

- The statistics in the `REQUEST_MISSES` column are consistently zero or static.
- The statistics in the `FREE_SPACE` column are more than 50 percent of the total size allocated to the reserved area.
Non-zero or steadily increasing values for the statistic `REQUEST_FAILURES` indicate that the reserved area is too small.

Oracle recommends that you aim to have `REQUEST_MISSES` and `REQUEST_FAILURES` near zero in `V$SHARED_POOL_RESERVED` when using the Shared Pool Reserved Area.

Another way to manage Library Cache space requests by large PL/SQL objects is through the use of the `ABORTED_REQUEST_THRESHOLD` procedure in the `DBMS_SHARED_POOL` package. This procedure is used to specify how much of the Library Cache’s memory can be flushed from the Shared Pool’s LRU list when a large PL/SQL object is trying to find space for itself in memory:

```
SQL> EXECUTE dbms_shared_pool.aborted_request_threshold (10000);
```

This example allows a large PL/SQL object to flush up to 10,000 bytes from the Shared Pool’s LRU list. The `ABORTED_REQUEST_THRESHOLD` limit is specified in bytes as a value between 5,000 and 2,000,000,000. If an application user exceeds this imposed limit, Oracle will return an out of memory error to the user.

**Keep Important PL/SQL Code in Memory**

Applications that make heavy use of PL/SQL packages and triggers or Oracle sequences can improve their Shared Pool hit ratios by permanently caching these frequently used PL/SQL objects in memory until the instance is shutdown. This process is known as *pinning* and is accomplished by using the Oracle-supplied PL/SQL package `DBMS_SHARED_POOL`. Pinned packages are stored in the space set aside for this purpose by the `init.ora` parameter `SHARED_POOL_RESERVED_SIZE`.

Even the command `ALTER SYSTEM FLUSH SHARED_POOL` does not flush pinned objects. However, pinned objects will be removed from memory at the next instance shutdown.
Building **DBMS_SHARED_POOL**

The DBMS_SHARED_POOL package is not built by the catproc.sql script that is run at database creation. Instead, it must be built by running the dbmspool.sql script located in $ORACLE_HOME/rdbms/admin on Unix systems and %ORACLE_HOME%/rdbms/admin on Windows 2000 systems while logged in as the user SYS:

```
SQL> @$ORACLE_HOME/rdbms/admin/dbmspool.sql
```

Using **DBMS_SHARED_POOL**

The DBMS_SHARED_POOL PL/SQL package contains two procedures, KEEP and UNKEEP, which are used to pin or unpinned (respectively) a package, trigger, or sequence in memory. The following example demonstrates how to pin an application PL/SQL package called APPROVE_PO:

```
SQL> EXECUTE DBMS_SHARED_POOL.KEEP ('APPROVE_PO');
```

Identifying Pinned Objects

You can identify the objects that have been pinned using DBMS_SHARED_POOL by issuing the following query on V$DB_OBJECT_CACHE:

```
SQL> SELECT owner, name, type
2  FROM v$db_object_cache
3  WHERE kept = 'YES';
```

<table>
<thead>
<tr>
<th>OWNER</th>
<th>NAME</th>
<th>TYPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>APPS</td>
<td>APPROVE_PO</td>
<td>PACKAGE</td>
</tr>
<tr>
<td>APPS</td>
<td>APPLY_TAX</td>
<td>TRIGGER</td>
</tr>
<tr>
<td>APPS</td>
<td>ASSIGN_ORG</td>
<td>PROCEDURE</td>
</tr>
</tbody>
</table>

What to Pin

Determining which objects should be pinned requires a thorough knowledge of the application and its components. One way to gather information on which objects are being frequently accessed in your database is to activate Oracle’s auditing feature on the objects that you are considering pinning. If you audit the PL/SQL packages, triggers, and sequences in the application for a period of normal processing, you can go back and see which of those objects were heavily accessed during that time and then consider pinning...
those items. Some vendors will also recommend pinning strategies for their products.

Straight SQL and anonymous blocks of PL/SQL code cannot be easily pinned using the `DBMS_SHARED_POOL` package. If you work in a development environment and your application issues straight SQL statements or anonymous blocks of PL/SQL code, you may be able to improve the application’s Library Cache hit ratio by rewriting these SQL statements as PL/SQL stored procedures or packages, which can then be pinned. You can identify the SQL statements contained in large, anonymous PL/SQL blocks by querying the `SQL_TEXT` column of the `V$SQLAREA` view for those SQL statements whose `COMMAND_TYPE` is 47 (i.e., PL/SQL Execute):

```
SQL> SELECT SUBSTR(sql_text,1,45), LENGTH(sql_text) 
  2  'STMT_SIZE' 
  3  FROM v$sqlarea 
  4  WHERE command_type = 47 
  5  ORDER BY LENGTH(sql_text) DESC;
```

In order to improve the Shared Pool’s hit ratios, Oracle recommends considering rewriting any anonymous PL/SQL block that exceeds 500 characters as a stored procedure or package instead.

**When to Pin**

To get the maximum benefit from pinning, the objects to be pinned should be loaded into memory right after instance startup. This is usually accomplished through the use of a SQL script that is manually executed immediately after the instance is started, or automatically upon instance startup by an `AFTER STARTUP ON DATABASE` database trigger. While some application vendors supply scripts for pinning their frequently used packages, many do not. In these cases, you will have to write a script yourself, consisting of an `EXECUTE DBMS_SHARED_POOL.KEEP` command for each package you wish to pin.

**Encourage Code Reuse**

Another way to improve the hit ratio for the Shared Pool is to use whenever possible the code that is already cached there. A new statement is only considered to be equivalent to a cached statement when the two statements
have the same hash value. To increase the likelihood that this will occur, it is important for application developers to use coding standards that specify the appropriate use of case, spacing, and lines in the application code. Use of generic code should also be considered where possible.

Another method to improve code reuse is through the use of bind variables. Bind variables allow application developers to use generic variables in their code rather than specific values. For example, suppose two users, User A and User B, both issue similar SQL statements. User A looks for Customer 2204:

```
SQL> SELECT last_name, first_name
       FROM app.customer
       WHERE customer_id = '2204';
```

User B looks for Customer 2301:

```
SQL> SELECT last_name, first_name
       FROM app.customer
       WHERE customer_id = '2301';
```

As you have seen earlier in this chapter, these two statements will not hash to the same value and will therefore cause a Library Cache cache miss. However, if these two statements had been issued using bind variables, User A looking for Customer 2204 using a bind variable would look like this:

```
:customer_id = 2204
SQL> SELECT last_name, first_name
       FROM app.customer
       WHERE customer_id = :customer_id;
```

User B looking for Customer 2301 using a bind variable would look like this:

```
:customer_id = 2301
SQL> SELECT last_name, first_name
       FROM app.customer
       WHERE customer_id = :customer_id;
```
In this case, the two statements will hash to the same value and result in a Library Cache cache hit because the bind variable itself, not the value stored in it, is used to generate the hash value.

The use of bind variables also has a tuning downside. Since bind variables are only containers for values, not actual values, they tend to confuse the cost-based optimizer, which is forced to make a blind guess at how the value in the container might affect the optimal execution plan.

Tune Library Cache–specific \texttt{init.ora} Parameters

In addition to \texttt{SHARED\_POOL\_SIZE}, there are three \texttt{init.ora} parameters that directly affect the performance on the Library Cache:

- \texttt{OPEN\_CURSORS}
- \texttt{CURSOR\_SPACE\_FOR\_TIME}
- \texttt{SESSION\_CACHED\_CURSORS}
- \texttt{CURSOR\_SHARING}

The following section discusses the purpose of each of these parameters and their impact on the performance of the Shared Pool.

\begin{table}[h]
\centering
\begin{tabular}{|l|p{0.7\textwidth}|}
\hline
\textbf{Oracle Objective} & Explain other tuning issues related to the Shared Pool \\
\hline
\end{tabular}
\end{table}

\textbf{OPEN\_CURSORS}

Each individual user session is assigned private SQL areas in the server’s main memory through the use of cursors. The \texttt{OPEN\_CURSORS} \texttt{init.ora} parameter limits how many of these cursors can be opened by a user’s session. The default value of 50 for this parameter is frequently too small for most applications. Increasing this value will not only allow each user to open more cursors, but also may minimize re-parsing of the user’s previously executed SQL statements, thus improving the Library Cache hit ratio.
**CURSOR_SPACE_FOR_TIME**

When this `init.ora` parameter is set to `TRUE`, shared SQL areas are pinned in the Shared Pool. This prevents the LRU mechanism from removing a shared SQL area from memory unless all cursors that reference that shared SQL area are closed. This can improve the Library Cache hit ratio and reduce SQL execution time, at the expense of using more of the server's main memory. The default value for `CURSOR_SPACE_FOR_TIME` is `FALSE`. If you have enough server memory to make the Shared Pool large enough to hold all the potential pinned SQL areas, and your application does not use frequent ad-hoc SQL, then consider setting this parameter to `TRUE`.

Oracle recommends that applications which utilize Oracle Forms should not set `CURSOR_SPACE_FOR_TIME` to `TRUE`.

**SESSION_CACHED_CURSORS**

When the same SQL statement is issued several times by the same user session, the `SESSION_CACHED_CURSORS` `init.ora` parameter can be used to specify how many of the cursors associated with these SQL statements should be cached in the Library Cache. The default value for this parameter is `0` (i.e., no cursor caching). If your application is form-based, setting this parameter to `TRUE` may improve the Library Cache hit ratio. This occurs because the SQL statements contained in the forms, which are repeatedly issued as users navigate from form to form, will be cached, thus minimizing statement re-parsing.

While all of the above parameters can improve the Library Cache hit ratio, each will generally require that the Shared Pool be increased in size, thus increasing the total server memory required to hold the SGA.

**CURSOR_SHARING**

The `CURSOR_SHARING` parameter in the `init.ora` allows you to change the Shared Pool's default behavior when parsing and caching SQL statements.
Table 4.4 describes the values that can be assigned to the CURSOR_SHARING parameter.

**TABLE 4.4 Values for CURSOR_SHARING Parameter**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FORCE</td>
<td>Allows two SQL statements, which differ only by a literal value, to share parsed code cached in the Shared Pool. The difference in the literal values must not change the meaning of the statement. This value was introduced in Oracle8i.</td>
</tr>
<tr>
<td>SIMILAR</td>
<td>Allows two SQL statements, which differ only by a literal value, to share parsed code cached in the Shared Pool. The difference in the literal values must not change the meaning of the statement or its cached execution plan. This value is new in Oracle9i.</td>
</tr>
<tr>
<td>EXACT</td>
<td>Two SQL statements must match exactly in order to share the parsed code cached in the Shared Pool. This is the default value for CURSOR_SHARING.</td>
</tr>
</tbody>
</table>

For example, suppose application user Bob issues the following SQL statement:

```sql
SQL> SELECT order_id, customer_id FROM orders WHERE order_id = 1001;
```

This SQL statement would be parsed and stored in the Shared Pool using the process described earlier in the chapter. Now, suppose application user Linda issues the following SQL statement:

```sql
SQL> SELECT order_id, customer_id FROM orders WHERE order_id = 1299;
```

Because these two statements are not identical, Linda would not have a cache hit on the Shared Pool and would instead have to parse her statement. However, these two statements are essentially identical with the exception of the literal values for `order_id` (i.e., 1001 in Bob’s query and 1299 in Linda’s query). By setting CURSOR_SHARING to a value of SIMILAR or FORCE, Oracle would consider these two similar SQL statements to be identical.
This would allow Linda to reuse Bob’s cached and parsed version of the SQL statement in the Shared Pool instead of parsing the statement again.

Oracle dynamically substitutes bind variables in place of literal values in order to achieve cursor sharing for similar statements.

Summary

The Shared Pool plays an important role in the processing of SQL statements issued against the instance by the application. It acts as the cache where previously issued statements are stored so subsequent requests for the same command can avoid re-parsing the same SQL command.

The Shared Pool caches its information in two areas: the Library Cache and the Data Dictionary Cache. The Library Cache is used to cache SQL statements; PL/SQL procedures, functions, and triggers; and sequences. The Data Dictionary Cache stores information about the objects in the database used by the cached SQL statements. Shared Pool performance is measured in terms of the hit ratios of these two caches. The hit ratios for these caches can be calculated by querying the contents of V$LIBRARYCACHE and V$ROWCACHE, respectively. The hit ratios can also be obtained from STATSPACK and REPORT.TXT output as well as Oracle Enterprise Manager’s Performance Manager graphs.

The cache-hit ratio for the Shared Pool can be improved by increasing the size of the Shared Pool, setting up a Shared Pool Reserved Area, pinning frequently used PL/SQL code and sequences in memory, by rewriting application SQL code to encourage the reuse of parsed statements, and by utilizing the OPEN_CURSORS, CURSOR_SPACE_FOR_TIME, and SESSION_CACHED_CURSORS init.ora parameters.

Exam Essentials

Understand each of the components of the Shared Pool. Know how each of these is used in application SQL processing and how the contents of each component is managed.
Know how to measure the performance of the Shared Pool and its components. Be able to query the Data Dictionary views, examine STATSPACK output, and examine REPORT.TXT files in order to find these performance indicators.

Understand what changes can be implemented in order to improve the performance of the Shared Pool and its components. Be able to explain which changes will have the greatest impact and which changes are more granular in nature.

Key Terms

Before you take the exam, be certain you are familiar with the following terms:

- bind variables
- cache hit
- cache miss
- Data Dictionary Cache
- hash values
- hit ratios
- Least Recently Used (LRU)
- Library Cache
- Parse Lock
- P-Code
- pinning
- Process Global Area (PGA)
- Shared Pool
- Shared Pool Reserved Area
- User Global Area
Review Questions

1. Which one of these is not part of the Shared Pool?
   A. Dictionary Cache
   B. Redo Log Buffer
   C. Library Cache
   D. User Global Area

2. The contents of the Shared Pool are managed using what type of algorithm?
   A. Least Recently Needed
   B. Least Recently Used
   C. Least Recently Accessed
   D. Least Recently Parsed

3. What is the process of preparing a statement for execution called?
   A. Caching
   B. Hashing
   C. Parsing
   D. None of the above

4. Finding a statement already cached in the Library Cache is referred to as what?
   A. Cache hit
   B. Cache miss
   C. Cache match
   D. Cache parse
5. To determine if a new statement is a match for an existing statement already in the Shared Pool, Oracle compares each statement’s ________________.
   A. Result set
   B. Security
   C. Execution plan
   D. Hashed value

6. In order for two statements to result in a cache hit, which of the following must be true?
   A. The statements must use the same case—uppercase, lowercase, or mixed.
   B. The statements must be issued against the same table.
   C. The statements must be on the same number of lines.
   D. All of the above must be true.

7. Which of the following dynamic Data Dictionary views contains the entire SQL statement issued by a user?
   A. V$LIBRARYCACHE
   B. V$SQLTEXT
   C. V$SQLAREA
   D. V$ROWCACHE

8. Oracle retains all of the following information for SQL statements in the Shared Pool except:
   A. The actual text of the statement itself
   B. The execution plan for the statement
   C. The P-Code of the statement
   D. Statistics associated with the statement
   E. All of the above is retained in the cache
9. Which dynamic Data Dictionary view contains information about the Library Cache hit ratio?
   A. V$ROWCACHE
   B. V$LIBRARYCACHE
   C. V$DICTIONARYCACHE
   D. All of the above

10. According to Oracle, what value for GETHITRATIO in V$LIBRARYCACHE should you strive for on an OLTP system?
   A. Less than 10 percent
   B. Between 60 and 80 percent
   C. Higher than 95 percent
   D. Between 10 and 20 percent

11. Changes in values for which of the following columns in V$LIBRARYCACHE indicates that a previously parsed statement was aged out of the cache, having to be subsequently re-parsed?
   A. INVALIDATIONS
   B. RELOADS
   C. PINHITRATIO
   D. GETHITRATIO

12. According to Oracle, what Reload Ratio would a well-tuned system have?
   A. 90 percent or higher
   B. 10 percent or higher
   C. Less than 1 percent
   D. More than 1 percent

13. If a sequence is altered to increment by a new value, each of the cached SQL statements that referenced this sequence will be ______________.
Review Questions 239

A. pinned in the Library Cache
B. marked as invalid in the Library Cache
C. hashed to a new value in the Library Cache
D. of no consequence to the Library Cache

14. You make a change to a view on a production server and recompile it. What will happen to any cached SQL that is dependent on the view during the next execution of the SQL?
   A. An Invalidation
   B. A Recompilation
   C. A Reorganization
   D. A Resubmission

15. Which of the following dynamic performance views has information on the hit ratio for the Data Dictionary Cache?
   A. V$LIBRARYCACHE
   B. V$DICTIONARYCACHE
   C. V$ROWCACHE
   D. V$SQLAREA

16. According to Oracle, what should the Data Dictionary hit ratio be for a well-tuned OLTP system?
   A. More than 85 percent
   B. Less than 85 percent
   C. Between 50 and 60 percent
   D. None of the above

17. You issue the command ALTER SYSTEM FLUSH SHARED_POOL. What happens to pinned objects?
   A. They are flushed from the cache.
   B. They are marked as invalid.
   C. They are recompiled upon the next execution.
   D. Nothing happens.
18. Which of the following would help improve the Shared Pool hit ratio if the application issues many similar, but not identical, SQL statements?
   A. Decrease the \texttt{SHARED\_POOL\_SIZE} \texttt{init.ora} parameter
   B. Increase the \texttt{SHARED\_POOL\_RESERVED\_SIZE} \texttt{init.ora} parameter
   C. Pin PL/SQL packages in the Shared Pool
   D. Rewrite application code to make more use of bind variables

19. Which of the following Oracle-supplied PL/SQL packages is used to pin PL/SQL packages into the Shared Pool?
   A. \texttt{DBMS\_UTILITY}
   B. \texttt{DBMS\_SYSTEM}
   C. \texttt{DBMS\_SHARED\_POOL}
   D. \texttt{DBMS\_KEEP}

20. Which dynamic performance view can be used to monitor the effectiveness of the Shared Pool Reserved Area?
   A. \texttt{V\_LIBRARYCACHE}
   B. \texttt{V\_ROWCACHE}
   C. \texttt{V\_DB\_OBJECT\_CACHE}
   D. \texttt{V\_SHARED\_POOL\_RESERVED}

21. Which of the following Oracle Enterprise Manager components contains performance information related to the Shared Pool?
   A. DBA Studio
   B. Performance Manager
   C. Lock Manager
   D. Change Manager
Answers to Review Questions

1. B. The Redo Log Buffer is a separate component of the System Global Area (SGA) and is not part of the Shared Pool, which is also part of the SGA. The three parts of the Shared Pool are the Dictionary Cache, Library Cache, and in certain configurations the User Global Area.

2. B. The LRU mechanism for managing the Shared Pool ensures that the most recently used statements will remain in memory, while older statements are moved out.

3. C. The parse process involves checking the statement for syntactic correctness, object resolution (i.e., table names, column names, and data types), and determination of an execution plan.

4. A. Cache hits occur whenever an application finds a parsed version of a statement already cached in memory.

5. D. Once reduced to its ASCII equivalent, each Oracle statement is then passed through a hashing algorithm. This hash value is compared to the hashed values associated with the statements that are cached in the Library Cache to determine if a parse needs to take place.

6. D. Two statements must match exactly in order to be considered a cache hit.

7. B. V$SQLAREA only contains the first 80 characters of the SQL statement issued.

8. E. Oracle caches all of this information about an SQL statement.

9. B. The GETHITRATIo and PINHITRATIo columns of V$LIBRARYCACHE show the hit ratios for the Library Cache.

10. C. A well-tuned OLTP system should have a GETHITRATIo for the Library Cache in excess of 95 percent.

11. B. A non-zero value for the RELOADS column of V$LIBRARYCACHE indicates that statements are being re-parsed shortly after they are flushed from the Library Cache.

12. C. A Reload Ratio of less than 1 percent is considered appropriate for most systems.
13. B. If an object that is referenced by a cached SQL statement is altered or dropped, all cached statements that refer to this object will be marked as invalid and will require a re-parse at the next execution of the statement.

14. A. Invalidations of SQL that is cached in the shared pool occur when code is recompiled or changes are made on dependent objects, such as adding a column to a table or modifying a view.

15. C. The V$ROWCACHE view contains two columns, GETMISSES and GETS, that can be used to calculate the hit ratio for the Data Dictionary Cache.

16. A. A well-tuned OLTP system will have a hit ratio of 85 percent or higher.

17. D. Pinned objects are not flushed from the shared pool even when the alter system flush shared pool command is issued. To remove these objects, you can use the DBMS_SHARED_POOL.UNKEEP procedure.

18. D. Improved Shared Pool hit ratios will be achieved if literal values in application SQL are replaced with bind variables because the number of cache hits will increase.

19. C. The KEEP and UNKEEP procedures of the DBMS_SHARED_POOL package can be used to pin and unpin PL/SQL packages in the Shared Pool.

20. D. The statistics for FREE_SPACE, REQUEST_MISSES, and REQUEST_FAILURES found in the V$SHARED_POOL_RESERVED view can all be used to determine if the Shared Pool Reserved Area is sized appropriately.

21. B. The Performance Manager component of the OEM Diagnostics Pack contains several tools that can monitor Library and Dictionary Cache hit ratios.
Tuning the Database Buffer Cache

ORACLE9i PERFORMANCE TUNING EXAM OBJECTIVES COVERED IN THIS CHAPTER:

- Describe how the Buffer Cache is used by different Oracle processes
- Describe the tuning issues related to the Buffer Cache
- Monitor the use of the Buffer Cache, also the different pools within the Buffer Cache
- Implement dynamic SGA allocation
- Set the DB_CACHE_ADVICE parameter
- Create and size multiple buffer pools

Exam objectives are subject to change at any time without prior notice and at Oracle’s sole discretion. Please visit Oracle’s Certification website (http://www.oracle.com/education/certification/) for the current exam objectives listing.
Your examination of Oracle SGA tuning began by discussing the Shared Pool in Chapter 4. You have seen how finding cached statements in memory can greatly enhance performance. Another significant way to improve performance is to make sure that users not only find their SQL and PL/SQL statements cached in memory, but also their requested data as well. This is the role of the Database Buffer Cache. Knowing how the Database Buffer Cache’s mechanisms work, how to monitor its performance, and how to tune it when needed are all key elements of Oracle performance tuning.

This chapter will discuss each of these areas beginning with a review of the Buffer Cache architecture.

Understanding the Database Buffer Cache

The Database Buffer Cache, or Buffer Cache, is the portion of the SGA that caches copies of the data blocks of the segments the application users have been most recently accessing in the database. Each buffer in the Buffer Cache corresponds in size to, and holds the contents of, one database block. The blocks cached in the Buffer Cache may belong to any of the following segment types:

- Tables
- Indexes
- Clusters
- Large Object (LOB) segments
- LOB indexes
- Rollback segments
- Temporary segments
The usage of these buffers in the Database Buffer Cache is managed by a combination of mechanisms including:

- An LRU list
- User Server Processes
- Database Writer (DBW0) Background Process

Each of these mechanisms is discussed in detail in the following sections.

### Oracle Objectives

| Describe how the Buffer Cache is used by different Oracle processes |
| Describe the tuning issues related to the Buffer Cache |

### The LRU List

When an application user issues an SQL statement, the segment data associated with that SQL statement will be copied from disk into the Buffer Cache in the SGA. For example, when a user issues a SELECT statement, the rows that satisfy the query must be read from disk into the Buffer Cache before they can be returned to the user. It is each individual user’s Server Process that is responsible for copying this data from the database datafiles into the buffers in the Buffer Cache.

Like the Shared Pool, the Database Buffer Cache is also managed by a Least Recently Used (LRU) algorithm. Once the Buffer Cache’s buffers are full, the LRU algorithm makes room for subsequent requests for new buffers by aging out the buffers that have been least recently accessed. This allows the Oracle Server to keep the most recently requested data buffers in the Buffer Cache while the less commonly used data buffers are flushed. By caching the data buffers in this manner, an application user who needs the same data buffers as a previous user benefits from the fact that the buffers are already in memory and therefore do not need to be read from disk by the user’s Server Process. This is beneficial because reads from memory are thousands of times faster than reads from disk.

You can think of the LRU list that manages the buffers in the Database Buffer Cache as being similar to a conveyor belt. Figure 5.1 illustrates this concept.
When a request is made for data, the user’s Server Process copies the blocks (A, B, C) from disk into buffers at the beginning (Most Recently Used) of the LRU List. These buffers stay on the LRU List but move toward the least recently used end as additional data blocks are read into memory. If a buffer (A) is accessed again during the time it is on the LRU List, the buffer is moved back to the beginning of the LRU List (i.e., back to the MRU end). If buffers are not accessed again before they reach the end of the LRU List, the buffers (B and C) are marked as free and will subsequently be overwritten by another block being read from disk.

The behavior of the LRU List is slightly different when a table is accessed during a full table scan (FTS). During an FTS, the table buffers are placed immediately at the least recently used end of the LRU List. While this has the effect of immediately causing these blocks to be flushed from the cache, it also prevents an FTS on a large table from pushing all other buffers out of the cache.

The buffers that the LRU list maintains can be in one of four states. These states are shown in Table 5.1.
Initially, the copies of the database blocks that are stored in free buffers contain the same data as their corresponding blocks on disk. In other words, the blocks and their copies in the buffers are in sync. Dirty buffers are those buffers whose copy in the Buffer Cache does not match the version of the block on disk. These blocks and their copies in the buffers are not in sync. While the Oracle Server can freely overwrite the contents of a free buffer when a new block needs to be placed in memory, dirty buffers cannot be overwritten by new blocks being read from disk until after the contents of the dirty buffers have been written to disk.

The mechanism for managing these dirty buffers is called the Dirty List or Write List. Using a checkpoint queue, this list keeps track of all the blocks that have been modified by application users through the use of INSERT, DELETE, or UPDATE statements, but have not yet been written to disk. Dirty buffers are written to disk by the Oracle background process Database Writer (DBW0). Each user’s Server Process also makes use of the Dirty List during while carrying out its Buffer Cache-related activities.

**User Server Process**

When a user’s Server Process does not find the data it needs already cached in the Buffer Cache, it must read the data from disk. Before the data can be read from disk, however, a free buffer must be found to hold the copy of the data block in memory. When searching for a free buffer to hold the

<table>
<thead>
<tr>
<th>State</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Free</td>
<td>Buffer is not currently in use.</td>
</tr>
<tr>
<td>Pinned</td>
<td>Buffer is currently in use by a Server Process.</td>
</tr>
<tr>
<td>Clean</td>
<td>A buffer that was just released from a pin, and has been marked for immediate reuse if the contents are not requested again because the data in the buffer was not changed.</td>
</tr>
<tr>
<td>Dirty</td>
<td>Buffer is not in use, but contains committed data that has not yet been written to disk by Database Writer.</td>
</tr>
</tbody>
</table>

**TABLE 5.1** Possible States for Database Buffer Cache Buffers
block that is to be read, the user’s Server Process makes use of both the LRU List and the Dirty List:

- While searching the LRU List for a free block, the Server Process moves any dirty blocks it finds on the LRU List to the Dirty List.
- As the Dirty List has buffers added to it, it grows longer and longer. When it hits a predetermined threshold length, DBW0 writes the dirty buffers to disk.
- DBW0 can also write dirty buffers to disk even if the Dirty List has not reached its threshold length. This occurs when a user’s Server Process has examined too many buffers without successfully finding a free buffer. In this case, DBW0 will write dirty buffers directly from the LRU List (as opposed to moving them to the Dirty List first, as above) to disk.

The Free Buffer Inspected statistic is the indicator of the number of buffers each Server Process is examining while in the process of finding a free buffer. This statistic is discussed in the section titled “Non–Hit Ratio Performance Measures” later in this chapter.

When a user’s Server Process needs a buffer and finds that buffer in memory, the buffer may contain uncommitted data or data that has changed since the time the user’s transaction began. Since the Oracle Server never allows a user to see uncommitted data or committed data that was changed after their transaction began, the Server Process uses a buffer that contains the before-image of the data to provide a read-consistent view of the database. This before-image is stored in a rollback segment whose blocks are also cached in the Buffer Cache.

The Database Writer background process also directly assists in the management of the LRU and Dirty Lists.

**Database Writer**

Not only can a user’s Server Process cause DBW0 to write dirty buffers to disk, there are several other database events that can cause Database Writer to write the dirty buffers to disk. The complete list of events that can cause DBW0 to write to disk is summarized in Table 5.2.
Measuring the Performance of the Database Buffer Cache

Now that you understand how the Database Buffer Cache is managed, you can begin to examine the ways it which its performance can be measured.

**TABLE 5.2** Events That Cause Database Writer to Write to Disk

<table>
<thead>
<tr>
<th>Event</th>
<th>How DBW0 Writes</th>
</tr>
</thead>
<tbody>
<tr>
<td>When Dirty List reaches its threshold length</td>
<td>DBW0 writes dirty buffers from Dirty List.</td>
</tr>
<tr>
<td>When LRU List is searched too long without finding a free buffer</td>
<td>DBW0 writes dirty buffers directly from the LRU List.</td>
</tr>
<tr>
<td>When three seconds pass</td>
<td>DBW0 moves dirty buffers from the LRU List to the Dirty List. If the threshold length is exceeded, a write to disk also occurs.</td>
</tr>
<tr>
<td>At Checkpoint</td>
<td>DBW0 moves all dirty buffers from the LRU List to the Dirty List and then writes them to disk.</td>
</tr>
<tr>
<td>At Database Shutdown</td>
<td>Unless a SHUTDOWN ABORT is used, DBW0 always writes all dirty buffers to disk at database shutdown.</td>
</tr>
<tr>
<td>At Tablespace Hot Backup</td>
<td>DBW0 moves the dirty buffers for that tablespace from the LRU List to the Dirty List and then writes them to disk.</td>
</tr>
<tr>
<td>At Tablespace Offline Temporary</td>
<td>DBW0 moves the dirty buffers for that tablespace from the LRU List to the Dirty List and then writes them to disk.</td>
</tr>
<tr>
<td>At Drop Segment</td>
<td>Dropping a table or index causes DBW0 to first write the dirty blocks for that segment to disk.</td>
</tr>
</tbody>
</table>

Now that you understand how the Database Buffer Cache is managed, you can begin to examine the ways it which its performance can be measured.

**Measuring the Performance of the Database Buffer Cache**

Like the Shared Pool, one indicator of the performance of the Database Buffer Cache is the cache *hit ratio*. Cache hits occur whenever a user finds that a data buffer needed by their SQL statement is already cached in memory;
cache misses occur when the user does not find the requested data to already be cached in memory—causing the data to be read from disk instead. High cache hit ratios indicate that your application users are frequently finding that the data buffers they need are already in memory. The hit ratio information for the Database Buffer Cache can be found in several places:

- V$SYSSTAT
- STATSPACK output
- UTLBSTAT.SQL/UTLESTAT.SQL output (REPORT.TXT)

Additionally, there are also non-hit ratio measures of Database Buffer Cache effectiveness, including:

- Free Buffer Inspected waits
- Buffer Busy Wait events
- Free Buffer Wait events

Oracle Enterprise Manager also contains graphical tools that contain useful information about Buffer Cache performance. The following sections discuss each of these measures of Buffer Cache performance.

### Oracle Objective

Monitor the use of the Buffer Cache, also the different pools within the Buffer Cache

### Monitoring the Buffer Cache Using V$SYSSTAT

The V$SYSSTAT dynamic performance view contains statistics regarding overall system performance, gathered since instance startup. The names of the four columns contained in the V$SYSSTAT view, along with descriptions of their contents, are shown in Table 5.3.

#### Table 5.3 Contents of V$SYSSTAT

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STATISTIC#</td>
<td>The identifier associated with the name of the performance statistic.</td>
</tr>
<tr>
<td>NAME</td>
<td>The name of the performance statistic.</td>
</tr>
</tbody>
</table>
There are over 200 different statistics tracked by the V$SYSSTAT view. However, only four of these are used when calculating the performance of the Database Buffer Cache:

**Physical Reads**  This statistic indicates the number of data blocks (i.e., tables, indexes, and rollback segments) read from disk into the Buffer Cache since instance startup.

**Physical Reads Direct**  This statistic indicates the number of reads that bypassed the Buffer Cache because the data blocks were read directly from disk instead. Because direct physical reads are done intentionally by Oracle when using certain features like Parallel Query, these reads are subtracted from the *Physical Reads* value when the Buffer Cache hit ratio is calculated. Otherwise, including these direct reads in the Buffer Cache hit ratio calculation would result in an artificially low hit ratio.

**Physical Reads Direct (LOB)**  This statistic indicates the number of reads that bypassed the Buffer Cache because the data blocks were associated with a Large Object (LOB) datatype.

Because direct physical reads are done intentionally by Oracle when accessing segments that contain LOB datatypes, these reads are also subtracted from the *Physical Reads* value when the Buffer Cache hit ratio is calculated.

---

**TABLE 5.3** Contents of V$SYSSTAT (continued)

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLASS</td>
<td>The broad category the statistic falls into, designated by a numeric code where: 1 = User, 2 = Redo, 4 = Enqueue, 8 = Cache, 16 = Operating System, 32 = Real Application Cluster, 64 = SQL, 128 = Debug. These values are additive, allowing a statistic to fall into more than one class.</td>
</tr>
<tr>
<td>VALUE</td>
<td>The numeric value currently associated with this statistic.</td>
</tr>
</tbody>
</table>
is calculated. Including these direct reads in the Buffer Cache hit ratio
calculation would also result in an artificially low hit ratio.

**Session Logical Reads** This statistic indicates the number of times a
request for a data block was satisfied by using a buffer that was already
cached in the Database Buffer cache. For read consistency, some of these
buffers may have contained data from rollback segments.

Figure 5.2 shows how to use these statistics to calculate the hit ratio for the
Database Buffer Cache.

**FIGURE 5.2** Calculating the Database Buffer Cache hit ratio using **V$SYSSTAT**

```
SQL> SELECT 1 - ((physical.value - direct.value - lob.value) / logical.value) 'Buffer Cache Hit Ratio'
2 FROM v$sysstat physical,
3     v$sysstat direct,
4     v$sysstat lob,
5     v$sysstat logical
6 WHERE physical.name = 'physical reads'
7 AND direct.name = 'physical reads direct'
8 AND lob.name = 'physical reads direct (lob)'
9 AND logical.name = 'session logical reads';
```

The results from the query in Figure 5.2 indicate that 93.7 percent of
the time, application users found that the data they were requesting was
already cached in the Buffer Cache. The general formula used in Figure 5.2
to calculate the Database Buffer Cache is:

\[
1 - \left( \frac{\text{Physical Reads} - \text{Physical Reads Direct} - \text{Physical Reads Direct [LOB]}}{\text{Session Logical Reads}} \right)
\]

According to Oracle, a well-tuned OLTP system should have Database Buffer
Cache hit ratios of 90 percent or higher.

**Monitoring the Buffer Cache Using STATSPACK**

The output from the STATSPACK utility also contains a calculation for the
Database Buffer Cache hit ratio. Figure 5.3 shows the appropriate section
from sample STATSPACK output with the Buffer Cache hit ratio highlighted
in bold.

The results from the sample output show that 92.21 percent of the time,
application users found that the data they were requesting was already
cached in the Buffer Cache.
Measuring the Performance of the Database Buffer Cache

FIGURE 5.3  STATSPACK output showing Buffer Cache hit ratio

<table>
<thead>
<tr>
<th>Instance Efficiency Percentages (Target 100%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Buffer NoWait %: 98.00</td>
</tr>
<tr>
<td>Buffer Hit %: 92.21</td>
</tr>
<tr>
<td>Library Hit %: 86.11</td>
</tr>
<tr>
<td>Execute to Parse %: 42.96</td>
</tr>
<tr>
<td>Parse CPU to Parse Elapsed %:</td>
</tr>
</tbody>
</table>

Using REPORT.TXT Output

The REPORT.TXT file generated by running UTLBSTAT.SQL and UTLESTAT.SQL also contains a section that shows the statistics needed to calculate the Database Buffer Cache hit ratio. These statistics include the same Physical Reads, Physical Reads Direct, Physical Reads Direct (LOB), and Session Logical Reads statistics that were queried from V$SYSSTAT in Figure 5.2. The relevant sections of a sample REPORT.TXT are shown in Figure 5.4.

FIGURE 5.4  REPORT.TXT output showing Buffer Cache statistics

```sql
SQL> select ni.name "Statistic",
2       ni.change "Total",
3       round(ni.change/trans.change,2) "Per Transaction",
4       round(ni.change/(ni.start_users + ni.end_users)/2,2) "Per Logon",
5       round(ni.change/(ni.to_number(ni.end_time, 'J') - ni.to_number(ni.start_time, 'J'))/60/60/24 - 1, 2) "Per Second"
6       from
7       statsdata ni,
8       statsdata trans,
9       where
10      trans.name='user commits'
11      and ni.change > 0
12      order by ni.name;
```

<table>
<thead>
<tr>
<th>Statistic</th>
<th>Total Per Transaction</th>
<th>Per Logon</th>
<th>Per Second</th>
</tr>
</thead>
<tbody>
<tr>
<td>physical reads</td>
<td>4032</td>
<td>12.33</td>
<td>25.04</td>
</tr>
<tr>
<td>physical reads direct</td>
<td>296</td>
<td>1.91</td>
<td>25.75</td>
</tr>
<tr>
<td>physical writes</td>
<td>199950</td>
<td>184.72</td>
<td>2498.75</td>
</tr>
<tr>
<td>physical writes direct</td>
<td>296</td>
<td>1.91</td>
<td>25.75</td>
</tr>
<tr>
<td>physical writes direct (lob)</td>
<td>97</td>
<td>0.43</td>
<td>6.22</td>
</tr>
<tr>
<td>physical writes non checkp</td>
<td>65902</td>
<td>60.26</td>
<td>810.5</td>
</tr>
<tr>
<td>physical writes non checkp</td>
<td>3107</td>
<td>28.77</td>
<td>388.38</td>
</tr>
<tr>
<td>process last non-idle time</td>
<td>5033083984</td>
<td>46653619.48</td>
<td>639185908</td>
</tr>
</tbody>
</table>

```sql
<map>
| recursive calls        | 170049                | 1574.53   | 21256.13   |
| rows fetched via callback | 10333            | 168.73    | 2277.68    |
| session connect time   | 5033083994           | 46652629.48 | 636159408 | 24197519.15 |
| session logical reads  | 764252                | 6520.85   | 80631.5    |
| session non memory     | 55372                 | 626.9     | 8717.5     |
```

14.22
Using these statistics, the Database Buffer Cache hit ratio would be calculated as:

\[
1 - \left(\frac{4032 - 206 - 22}{704252}\right) = 0.9945985
\]

or a 99.46 percent hit ratio.

**Non-Hit Ratio Performance Measures**

Hit ratios are one effective measure of Buffer Cache performance. However, there are also several non-hit ratio measures of Database Buffer Cache performance. These indicators are Buffer Cache-related system statistics found in dynamic performance views and the output from STATSPACK and REPORT.TXT.

Because these statistics can help guide you toward the true tuning trouble-spot, using them as a starting point for Buffer Cache tuning is often more effective than calculating Buffer Cache hit ratios and blindly increasing the size of the Buffer Cache.

The following system statistics are related to the process of finding a free buffer in the Database Buffer Cache. There are three statistics that measure this activity:

- **Free Buffer Inspected** Number of Buffer Cache buffers inspected by user Server Processes before finding a free buffer. A closely related statistic is **dirty buffer inspected**, which represents the total number of dirty buffers a user process found while trying to find a free buffer.

- **Free Buffer Waits** Number of waits experienced by user Server Processes during Free Buffer Inspected activity. These waits occur whenever the Server Process had to wait for Database Writer to write a dirty buffer to disk.

- **Buffer Busy Waits** Number of times user Server Processes waited for a free buffer to become available. These waits occur whenever a buffer requested by user Server Processes is already in memory, but is in use by another process. These waits can occur for rollback segment buffers as well as data and index buffers.

High or steadily increasing values for any of these statistics indicate that user Server Processes are spending too much time searching for, and waiting for access to, free buffers in the Database Buffer Cache.

The V$SYSSTAT and V$SYSTEM_EVENT dynamic performance views can be used to monitor these secondary indicators of Buffer Cache Performance.
The `V$SYSSTAT` view contains statistics on the Free Buffer Inspected statistic and the `V$SYSTEM_EVENT` view contains statistics on the Free Buffer Waits and Buffer Busy Waits. All three of these statistics can be combined into one query using the SQL statement shown below:

```sql
SQL> SELECT name, value
2  FROM v$sysstat
3  WHERE name IN ('free buffer inspected')
4  UNION
5  SELECT event, total_waits
6  FROM v$system_event
7  WHERE event in ('free buffer waits','buffer busy waits');
```

<table>
<thead>
<tr>
<th>NAME</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>buffer busy waits</td>
<td>170</td>
</tr>
<tr>
<td>free buffer inspected</td>
<td>0</td>
</tr>
</tbody>
</table>

These statistics are also included in the output from STATSPACK and `REPORT.TXT`. Sample STATSPACK output showing these statistics can be seen in Figures 5.5 and 5.6.

**FIGURE 5.5** STATSPACK output showing Free Buffer Inspected statistics

```

<table>
<thead>
<tr>
<th>Statistic</th>
<th>Total</th>
<th>per Second</th>
<th>per Trans</th>
</tr>
</thead>
<tbody>
<tr>
<td>enqueue waits</td>
<td>6</td>
<td>0.0</td>
<td>9.0</td>
</tr>
<tr>
<td>execute count</td>
<td>542</td>
<td>0.2</td>
<td>47.5</td>
</tr>
<tr>
<td>free buffer inspected</td>
<td>41</td>
<td>0.1</td>
<td>13.7</td>
</tr>
<tr>
<td>free buffer requested</td>
<td>704</td>
<td>1.1</td>
<td>334.7</td>
</tr>
<tr>
<td>immediate (CP) block cleanup app</td>
<td>0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>immediate (PENDING) block cleanup</td>
<td>16</td>
<td>0.0</td>
<td>5.4</td>
</tr>
</tbody>
</table>
```

**FIGURE 5.6** STATSPACK output showing Buffer Busy Wait statistics

```

-> Standard block size Pools D: default, K: keep, R: recycle
   -> Default Pools for other block sizes: 2K, 4K, 8K, 16K, 32K

<table>
<thead>
<tr>
<th>P</th>
<th>Number of Pools</th>
<th>Buffers Hi %</th>
<th>Buffers Lo %</th>
<th>Buffers Reads</th>
<th>Buffers Writes</th>
<th>Buffers Complete</th>
<th>Buffers Busy</th>
</tr>
</thead>
<tbody>
<tr>
<td>D</td>
<td>6,018</td>
<td>100.0</td>
<td></td>
<td>3,165</td>
<td>0</td>
<td>236</td>
<td>2</td>
</tr>
</tbody>
</table>
```
Tuning the Database Buffer Cache

Sample REPORT.TXT output showing these statistics can be seen in Figures 5.7 and 5.8.

**FIGURE 5.7** REPORT.TXT output showing Free Buffer Inspected statistics

```sql
SELECT nl.name "Statistic", 
       round(nl.change/transactions) "Per Transaction", 
       round(nl.change/((start_users + end_users)/2)) "Per Logon", 
       round(nl.change/to_number(to_char(start_time, 'DD-MON-RR'))*60*60*24) "Per Second" 
FROM statsstats nl, 
     statsstats trans, 
     statsstats db 
WHERE trans.name='user commits' 
AND nl.change != 0 
ORDER BY nl.name;
```

<table>
<thead>
<tr>
<th>Statistic</th>
<th>Total Per Transact</th>
<th>Per Logon</th>
<th>Per Second</th>
</tr>
</thead>
<tbody>
<tr>
<td>enqueue timeouts</td>
<td>5</td>
<td>.05</td>
<td>.63</td>
</tr>
<tr>
<td>enqueue waits</td>
<td>9</td>
<td>.08</td>
<td>1.13</td>
</tr>
<tr>
<td>free buffer inspected</td>
<td>8052</td>
<td>745.39</td>
<td>10062.75</td>
</tr>
<tr>
<td>free buffer requested</td>
<td>17415</td>
<td>161.25</td>
<td>2176.80</td>
</tr>
<tr>
<td>immediate (cr) block clean</td>
<td>427</td>
<td>4.95</td>
<td>53.39</td>
</tr>
</tbody>
</table>

**FIGURE 5.8** REPORT.TXT output showing Buffer Busy Wait statistics

```sql
SQL> Rem System wide wait events for non-background processes (PMON, 
SQL> Rem DBA же в IPON, etc). Times are in hundreds of seconds. Each one of 
SQL> Rem these is a context switch which costs CPU time. By looking at 
SQL> Rem the total time you can often determine what is the bottleneck. 
SQL> Rem that processes are waiting for. This shows the total time spent 
SQL> Rem waiting for a specific event and the average time per wait on 
SQL> Rem that event. 
SQL> SELECT nl.event "Event Name", 
            nl.event_count "Count", 
            nl.time_waited/"Total Time", 
            round(nl.time_waited/nl.event_count) "Avg Time" 
FROM statsevent nl 
WHERE nl.event_count > 0 
ORDER BY nl.time_waited desc;
```

<table>
<thead>
<tr>
<th>Event Name</th>
<th>Count</th>
<th>Total Time</th>
<th>Avg Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>enqueue message from client</td>
<td>40057</td>
<td>38572</td>
<td>.89</td>
</tr>
<tr>
<td>enqueue</td>
<td>21</td>
<td>4724</td>
<td>224.95</td>
</tr>
<tr>
<td>library cache load lock</td>
<td>18</td>
<td>2487</td>
<td>132.72</td>
</tr>
<tr>
<td>log file switch completion</td>
<td>39</td>
<td>2672</td>
<td>68.51</td>
</tr>
<tr>
<td>db file sequential read</td>
<td>290</td>
<td>2614</td>
<td>6.98</td>
</tr>
<tr>
<td>db file sequential read</td>
<td>98</td>
<td>1476</td>
<td>15.09</td>
</tr>
<tr>
<td>db file read</td>
<td>99</td>
<td>1151</td>
<td>11.22</td>
</tr>
<tr>
<td>db file scattered read</td>
<td>450</td>
<td>1059</td>
<td>2.35</td>
</tr>
<tr>
<td>log buffer space</td>
<td>102</td>
<td>1011</td>
<td>6.24</td>
</tr>
<tr>
<td>switch log file command</td>
<td>5</td>
<td>681</td>
<td>136.2</td>
</tr>
<tr>
<td>buffer busy waits</td>
<td>170</td>
<td>246</td>
<td>1.45</td>
</tr>
<tr>
<td>control file sequential read</td>
<td>55</td>
<td>244</td>
<td>4.38</td>
</tr>
<tr>
<td>latch free</td>
<td>119</td>
<td>195</td>
<td>1.64</td>
</tr>
</tbody>
</table>
Using OEM to Monitor Buffer Cache Performance

The Performance Manager component of the OEM console can also be used to monitor Buffer Cache performance. Figure 5.9 shows an example of the Performance Manager output related to the Buffer Cache.

The output in Figure 5.9 shows that the Buffer Cache hit ratio was 97.38 percent during the OEM monitoring period.

FIGURE 5.9 OEM Performance Manager Buffer Cache monitor

Improving Buffer Cache Performance

Now that you know how the Database Buffer Cache management mechanisms work and how to monitor their performance, you can examine methods for improving the performance. The objective of these methods is to increase Database Buffer Cache performance by improving its hit ratio. These techniques fall into five categories:

- Make it bigger.
- Use multiple Buffer Pools.
- Cache tables in memory.
• Bypass the Buffer Cache.
• Use indexes appropriately.

Make It Bigger

The easiest way to improve the performance of the Database Buffer Cache is to increase its size. The larger the Database Buffer Cache, the less likely cached buffers are to be moved out of the cache by the LRU List. The longer buffers are stored in the Database Buffer Cache, the higher the hit ratio will be. Increasing the size of the Buffer Cache will also lower the number of waits reported by the free buffer inspected, buffer busy waits, and free buffer waits system statistics.

In Oracle9i, the size of the Database Buffer Cache is determined by the several init.ora parameters. A description of each of these parameters is shown in Table 5.4.

**TABLE 5.4** The *init.ora* Parameters Affecting Buffer Cache Size

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DB_BLOCK_SIZE</td>
<td>Defines the primary block size for the database. Oracle9i supports multiple block sizes, allowing the block size for a specific tablespace to vary from this size.</td>
</tr>
<tr>
<td>DB_CACHE_SIZE</td>
<td>Determines the size of the Default Buffer Pool for buffers with the primary block size as established by DB_BLOCK_SIZE. The default size is 48M.</td>
</tr>
<tr>
<td>DB_KEEP_CACHE_SIZE</td>
<td>Determines the size of the Keep Buffer Cache. The default size is 0KB. Also uses the primary block size as its buffer size. The Keep Cache is discussed in the next section, “Use Multiple Buffer Pools.”</td>
</tr>
<tr>
<td>DB_RECYCLE_CACHE_SIZE</td>
<td>Determines the size of the Recycle Buffer Cache. The default size is 0KB. Also uses the primary block size as its buffer size. The Recycle Cache is discussed in the next section, “Use Multiple Buffer Pools.”</td>
</tr>
<tr>
<td>DB_2K_CACHE_SIZE</td>
<td>If 2KB is not the primary block size, specifies the size of the Buffer Cache for segments with a 2KB block size.</td>
</tr>
</tbody>
</table>
The primary database block size is specified in bytes by the `DB_BLOCK_SIZE` parameter. This value is set at database creation and can only be changed by recreating the database. The default value for `DB_BLOCK_SIZE` varies by platform. On most Unix systems, it is between 2KB and 8KB. Windows 2000 systems have a default block size of 2KB. No matter the default, most systems can use 2KB, 4KB, 8KB, 16KB, or 32KB block sizes. Not all tablespaces need to have the same block size. The concept of granules, discussed in Chapter 2, “Sources of Tuning Information,” allows tablespaces within an Oracle9i database to have varying block sizes. However, two tablespaces, `SYSTEM` and `TEMP`, will always have the primary block size as defined by the `DB_BLOCK_SIZE` `init.ora` parameter, as will any other tablespace that was not explicitly created with a different block size. The Database Buffer Cache should be tuned in relation to this primary block size even though it can store buffers of various sizes.

Do not use the `DB_nK_CACHE_SIZE` parameter to define the size of the Buffer Cache for the primary block size. Always use the `DB_CACHE_SIZE` parameter to set the size of the Buffer Cache for the primary block size.

In order to improve performance, the Buffer Cache can be divided into up to three different areas called the Default, Keep, and Recycle pools using the `init.ora` parameters shown in Table 5.4. The size of each of these parameters can be expressed in kilobytes, megabytes, or gigabytes by appending the

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>DB_4K_CACHE_SIZE</code></td>
<td>If 4KB is not the primary block size, specifies the size of the Buffer Cache for segments with a 4KB block size.</td>
</tr>
<tr>
<td><code>DB_8K_CACHE_SIZE</code></td>
<td>If 8KB is not the primary block size, specifies the size of the Buffer Cache for segments with a 8KB block size.</td>
</tr>
<tr>
<td><code>DB_16K_CACHE_SIZE</code></td>
<td>If 16KB is not the primary block size, specifies the size of the Buffer Cache for segments with a 16KB block size.</td>
</tr>
<tr>
<td><code>DB_32K_CACHE_SIZE</code></td>
<td>If 32KB is not the primary block size, specifies the size of the Buffer Cache for segments with a 32KB block size.</td>
</tr>
</tbody>
</table>
appropriate character (K, M, or G) to the numeric value specified for the parameter. By default, only the Default Pool is configured, as both the Recycle and Keep Pool are assigned zero bytes. The use of multiple Buffer Pools is discussed in a subsequent section, “Use Multiple Buffer Pools.” In terms of performance, increasing the size of the Default Buffer Pool will generally improve the Buffer Cache hit ratio. The size of the Default Pool can be adjusted either dynamically or via an instance shutdown and startup.

### Oracle Objective

Implement dynamic SGA allocations

### Dynamically Increasing the Buffer Cache Size

The size of the Database Buffer Cache and its three component pools can all be changed dynamically using the ALTER SYSTEM command. Figure 5.10 shows the Default Pool being increased from 48MB to 100MB using the ALTER SYSTEM command.

**Figure 5.10** Dynamic increase of Default Pool size

```
SQL> show parameter db_cache_size
NAME                TYPE VALUE
------------------- -------- -------------------------
db_cache_size       unknown 50331648

SQL> ALTER SYSTEM SET db_cache_size = 100m;
System altered.

SQL> show parameter db_cache_size
NAME                TYPE VALUE
------------------- -------- -------------------------
db_cache_size       unknown 109681904
```

Changing the size of the Buffer Cache using the ALTER SYSTEM command will cause the instance’s associated SPFILE to also be updated. This will make the change effective even after the next instance restart.

When adjusting the sizes of the Database Buffer Cache pools, three rules must be kept in mind:

- The size specified must be a multiple of the granule size.
- The total size of the Buffer Cache, Shared Pool, and Redo Log Buffer cannot exceed the value specified by SGA_MAX_SIZE.
The Default Pool, designated by the init.ora parameter DB_CACHE_SIZE, cannot be set to a size of zero.

Attempting to increase the size of the Buffer Cache to a value greater than that of the available granules will cause the ORA-00384: Insufficient memory to grow cache error to occur.

Manually Increasing the Buffer Cache Size

The Buffer Cache can also be resized by shutting down the instance, changing the value for DB_CACHE_SIZE, and then restarting the instance. This technique is required whenever you need to increase the value of SGA_MAX_SIZE so that a larger value for DB_CACHE_SIZE can be specified.

Shutting down and then restarting the instance will produce artificially low Buffer Cache hit ratios until the instance “warms up” and the cache is reloaded with buffers from the database datafiles.

Increasing the Size of the Buffer Cache: How Much Is Enough?

In general, it is an acceptable tuning practice to keep making the Database Buffer Cache either dynamically or manually larger by increasing the value for DB_CACHE_SIZE until no further improvement in the Database Buffer Cache hit ratio occurs or until the available server memory is exhausted. However, instead of simply guessing at how much bigger to make the Buffer Cache, you can use the Oracle9i Buffer Cache Advisory feature. The Buffer Cache Advisory feature tracks the changes that might occur in Buffer Cache performance if the Buffer Cache were made larger (or smaller) than its existing size. Using the Buffer Cache Advisory feature requires some initial configuration.

Oracle Objective

Set the DB_CACHE_ADVICE parameter

Configuring the Buffer Cache Advisory Feature

You must set the init.ora parameter DB_CACHE_ADVICE=ON in order for the Buffer Cache Advisory feature to gather statistics about how changes to the Buffer Cache size might affect performance. Setting this parameter to ON causes Oracle9i to allocate memory to the Buffer Cache Advisory
feature and causes the Buffer Cache Advisory feature to start gathering statistics about the Buffer Cache’s performance. Other possible values for the `DB_CACHE_ADVICE` init.ora parameter follow:

**OFF**  
Turns off the Buffer Cache Advisory feature and releases any memory or CPU resources allocated to it.

**READY**  
Pre-allocates memory to the Buffer Cache Advisory process, but does not actually start it. Therefore, no CPU is consumed and no statistics are gathered until the `DB_CACHE_ADVICE` parameter is set to ON. The READY option is used to ensure that the Buffer Cache Advisory feature will find the memory it requires when it is activated at a later time.

As with the `DB_CACHE_SIZE` parameter, the `DB_CACHE_ADVICE` parameter can also be set either manually or dynamically. In addition to memory, activating the Buffer Cache Advisory feature also consumes some CPU resources as well. Therefore, you may want to limit your use of this parameter to those times when you are undertaking a performing tuning effort.

If you did not previously use the READY option, you may get an ORA-04031 unable to allocate n bytes of shared memory error when you dynamically try to set `DB_CACHE_ADVICE` to ON.

In order to get reliable recommendations, you should wait 30 minutes or more after activating the Buffer Cache Advisory feature before examining the recommendation statistics.

**Examining the Buffer Cache Advisory Statistics**

The performance statistics gathered by the Buffer Cache Advisory feature are stored in the `V$DB_CACHE_ADVICE` dynamic performance view. The contents of this view are described in Table 5.5.

<table>
<thead>
<tr>
<th><strong>TABLE 5.5</strong> Description of <code>V$DB_CACHE_ADVICE</code> View</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Column Name</strong></td>
</tr>
<tr>
<td>ID</td>
</tr>
<tr>
<td>NAME</td>
</tr>
<tr>
<td>BLOCK_SIZE</td>
</tr>
</tbody>
</table>
After a period of normal database processing has passed, you can examine the `V$DB_CACHE_ADVICE` view to see what performance gains could be expected from the various `DB_CACHE_SIZE` values considered by the Buffer Cache Advisory feature. The following query shows sample output from a query on `V$DB_CACHE_ADVICE`, note that the query limits the output to those records with an 8KB (8,192 bytes) block size since this is the primary block size for the database as specified by the `DB_BLOCK_SIZE init.ora` parameter:

```
SQL> SELECT name, size_for_estimate, estd_physical_reads
2  FROM v$db_cache_advice
3  WHERE block_size = '8192'
4  AND advice_status = 'ON';
```

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADVICE_STATUS</td>
<td>Status of the Buffer Cache Advisory feature (i.e., OFF, READY, ON).</td>
</tr>
<tr>
<td>SIZE_FOR_ESTIMATE</td>
<td>Cache size estimate used when predicting the number of physicals reads, expressed in MB.</td>
</tr>
<tr>
<td>BUFFERS_FOR_ESTIMATE</td>
<td>Cache size estimate used when predicting the number of physicals reads, expressed in buffers.</td>
</tr>
<tr>
<td>ESTD_PHYSICAL_READ_FACTOR</td>
<td>Ratio comparing the estimated number of physical reads to the actual number of reads that occurred against the current cache, null if no physical reads occurred.</td>
</tr>
<tr>
<td>ESTD_PHYSICAL_READS</td>
<td>Estimated number of physical reads that would have occurred if the cache size specified by SIZE_FOR_ESTIMATE and BUFFER_FOR_ESTIMATE had actually been used for the DB_CACHE_SIZE.</td>
</tr>
</tbody>
</table>

After a period of normal database processing has passed, you can examine the `V$DB_CACHE_ADVICE` view to see what performance gains could be expected from the various `DB_CACHE_SIZE` values considered by the Buffer Cache Advisory feature. The following query shows sample output from a query on `V$DB_CACHE_ADVICE`, note that the query limits the output to those records with an 8KB (8,192 bytes) block size since this is the primary block size for the database as specified by the `DB_BLOCK_SIZE init.ora` parameter:
Tuning the Database Buffer Cache

<table>
<thead>
<tr>
<th>NAME</th>
<th>SIZE_FOR_ESTIMATE</th>
<th>ESTD_PHYSICAL_READS</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEFAULT</td>
<td>10.1797</td>
<td>7686</td>
</tr>
<tr>
<td>DEFAULT</td>
<td>20.3594</td>
<td>6614</td>
</tr>
<tr>
<td>DEFAULT</td>
<td>30.5391</td>
<td>6028</td>
</tr>
<tr>
<td>DEFAULT</td>
<td>40.7188</td>
<td>5785</td>
</tr>
<tr>
<td>DEFAULT</td>
<td>50.8984</td>
<td>5497</td>
</tr>
<tr>
<td>DEFAULT</td>
<td>61.0781</td>
<td>5342</td>
</tr>
<tr>
<td>DEFAULT</td>
<td>71.2578</td>
<td>4915</td>
</tr>
<tr>
<td>DEFAULT</td>
<td>81.4375</td>
<td>4867</td>
</tr>
<tr>
<td>DEFAULT</td>
<td>91.6172</td>
<td>4866</td>
</tr>
<tr>
<td>DEFAULT</td>
<td>101.7969</td>
<td>4866</td>
</tr>
<tr>
<td>DEFAULT</td>
<td>111.9766</td>
<td>4866</td>
</tr>
<tr>
<td>DEFAULT</td>
<td>122.1563</td>
<td>4866</td>
</tr>
<tr>
<td>DEFAULT</td>
<td>132.3359</td>
<td>4866</td>
</tr>
<tr>
<td>DEFAULT</td>
<td>142.5156</td>
<td>4866</td>
</tr>
<tr>
<td>DEFAULT</td>
<td>152.6953</td>
<td>4866</td>
</tr>
<tr>
<td>DEFAULT</td>
<td>162.8753</td>
<td>4866</td>
</tr>
<tr>
<td>DEFAULT</td>
<td>173.0547</td>
<td>4866</td>
</tr>
<tr>
<td>DEFAULT</td>
<td>183.2344</td>
<td>4866</td>
</tr>
<tr>
<td>DEFAULT</td>
<td>193.4141</td>
<td>4866</td>
</tr>
<tr>
<td>DEFAULT</td>
<td>203.5938</td>
<td>4866</td>
</tr>
</tbody>
</table>

Assume the instance that generated this output currently has a value of 30MB for DB_CACHE_SIZE. This value appears three rows down in the output, as a value of 30.5391 under the SIZE_FOR_ESTIMATE column. The ESTD_PHYSICAL_READS column indicates that, based on the database activity that occurred during the period when the Buffer Cache Advisory feature was enabled, 6028 physical reads can be expected when the DB_CACHE_SIZE is this size.

If the DB_CACHE_SIZE were reduced to 20MB, you could expect 6614 physical reads given the same amount of database activity. Likewise, if the Buffer Cache were increased to 100MB, you could expect 4866 physical reads given the same amount of database activity.

In this way, the Buffer Cache Advisory feature gives you an “apples to apples” comparison of how many additional, or how many fewer, physical I/Os can be expected given an increase or decrease in the size of the DB_CACHE_SIZE init.ora parameter.
The query output shows that the number of physical I/Os is not expected to be significantly reduced once the `DB_CACHE_SIZE` is increased to 80MB. The increase from 80MB to 90MB would only reduce physical I/Os by 2, and increases beyond 90MB are estimated to produce no fewer physical I/Os at all.

Increases to the Database Buffer Cache are not generally a direct relationship. For example, doubling the value for `DB_CACHE_SIZE` will not necessarily double the Buffer Cache hit ratio.

Another technique for improving performance of the Buffer Cache is to break up the Buffer Cache into multiple Buffer Pools.

**Use Multiple Buffer Pools**

By default, all database segments compete for use of the same Database Buffer Cache buffers in the Default Pool. This can lead to a situation where some infrequently accessed tables that are only of interest to a few application users can push more-frequently-used data buffers out of memory. To avoid this, Oracle provides you with the ability to divide the Database Buffer Cache into as many as three separate areas called Buffer Pools. Segments are then explicitly assigned to use the appropriate Buffer Pool as determined by the DBA.

**Buffer Pool Types**

Before you create any Buffer Pools, you must determine which types of pools you wish to use. Each Buffer Pool is used to cache segments based on how frequently you expect those segments to be accessed by the application:

- **Keep Pool**  Used to cache segments that you rarely, if ever, want to leave the Database Buffer Cache. The size of this pool is designated in bytes, kilobytes, megabytes, or gigabytes, by the `init.ora` parameter `DB_KEEP_CACHE_SIZE`. The use of the Keep Pool is optional. Its default size is zero bytes.

- **Recycle Pool**  Used to cache segments that you rarely, if ever, want to retain in the Database Buffer Cache. The size of a Recycle Pool is designated in bytes, kilobytes, megabytes, or gigabytes, by the `init.ora` parameter `DB_RECYCLE_CACHE_SIZE`. The use of a Recycle Pool is also optional. Its default size is zero bytes.
Default Pool  Used to cache segments that are not designated for either
the Keep or Recycle pools. The size of this pool is designated in bytes, kilo-
bytes, megabytes, or gigabytes, by the init.ora parameter DB_CACHE_-
SIZE. The Default Pool is required. Its default size is 48 megabytes. This
parameter cannot be set to zero.

Unless you specify otherwise in the init.ora, only the Default Pool is
created at instance startup.

If multiple block sizes are being used, a Keep, Recycle, and Default Pool can
be created for each block size’s Buffer Cache.

Unlike Oracle8i, where the memory for the Keep and Recycle pools was taken
from the memory allocated to the Default Pool, Oracle9i independently
assigns the memory to each of the three Buffer Pool types. This allows the size
of each of the Buffer Pools to be modified separately, without affecting the
memory assigned to the remaining pools.

Before the Buffer Pools can be configured, you will need to determine
which segments will be cached in the Keep and Recycle pools.

Determining Which Segments to Cache

Determining which segments to cache in a non-Default Buffer Pool, and
in which pools to cache them, requires a thorough knowledge of the applica-
tion, its segments, and how the segments are accessed. One way to determine
which segments should be assigned to each Buffer Pool is to determine which
segments are currently cached in the Default Pool and classify these segments
as possible candidates for inclusion in the Keep or Recycle pools. Four views,
V$BH, DBA_OBJECTS, V$CACHE, and DBA_USERS can be used for this purpose.

Using V$BH and DBA_OBJECTS to Identify Caching Candidates

The V$BH and DBA_OBJECTS views both contain useful information that can
help you identify which segments are currently cached in the Buffer Cache.
Using this information and your knowledge of the application, you can
determine whether any of these objects should be cached in the Keep or
Recycle Pools in an effort to improve the performance of the Buffer Cache.
Selected columns from these views that are useful for assigning segments to
appropriate Buffer Pools are described in Table 5.6 and Table 5.7.
The V$BH view is actually intended for use when tuning Oracle Servers running the Oracle9i Real Application Clusters feature (RAC). However, it can also be useful for tuning non-RAC systems because it displays how many cached buffers from each cached segment are currently in the Database Buffer Cache. By querying the V$BH and DBA_OBJECTS views using the common column OBJECT_ID, you can also see which objects are currently cached in the Buffer Cache; how many Buffer Cache buffers those objects are using; and owner, name, and type of those objects:

```sql
SQL> SELECT obj.owner,
          2      obj.object_name,
          3      obj.object_type,
          4      COUNT(DISTINCT bh.block#) 'Num. Buffers'
          5  FROM   dba_objects obj, v$bh bh
          6  WHERE  obj.object_id = bh.objd
          7  AND    owner != 'SYS'
          8  GROUP BY obj.owner,
          9          obj.object_name,
         10          obj.object_type
          11 ORDER BY 4;
```

**Table 5.6** Partial Description of V$BH View

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLOCK#</td>
<td>The block number of the block cached in the Buffer Cache</td>
</tr>
<tr>
<td>OBJD</td>
<td>The Object ID of the object associated with the blocks that are cached in the Buffer Cache</td>
</tr>
</tbody>
</table>

**Table 5.7** Partial Description of DBA_OBJECTS View

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OBJECT_ID</td>
<td>The numeric identifier assigned to the object</td>
</tr>
<tr>
<td>OWNER</td>
<td>The owner of the object</td>
</tr>
<tr>
<td>OBJECT_NAME</td>
<td>The name of the object</td>
</tr>
<tr>
<td>OBJECT_TYPE</td>
<td>The object type (e.g., table, index, view)</td>
</tr>
</tbody>
</table>

The V$BH view is actually intended for use when tuning Oracle Servers running the Oracle9i Real Application Clusters feature (RAC). However, it can also be useful for tuning non-RAC systems because it displays how many cached buffers from each cached segment are currently in the Database Buffer Cache. By querying the V$BH and DBA_OBJECTS views using the common column OBJECT_ID, you can also see which objects are currently cached in the Buffer Cache; how many Buffer Cache buffers those objects are using; and owner, name, and type of those objects:
Segments owned by the user SYS are excluded from this query because they are most likely data dictionary segments that are nearly always kept on the LRU List during normal processing.

Using the output from this query you can see that the EMPLOYEE and REGION tables along with the EMPLOYEE_PK_ID index are currently cached in memory. Depending on their usage patterns, these segments may make good candidates for the Keep or Recycle Pool. Another way to find similar information is by querying the V$CACHE and DBA_USERS views.

Using V$CACHE and DBA_USERS to Identify Caching Candidates

Two additional views, V$CACHE and DBA_USERS, can give additional information about segments that might make good candidates for caching in the Keep or Recycle Pools. Selected columns from these views that are useful for assigning segments to appropriate Buffer Pools are described in Table 5.8 and Table 5.9.

**TABLE 5.8** Partial Description of V$CACHE View

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLOCK#</td>
<td>The block number of the block cached in the Buffer Cache</td>
</tr>
<tr>
<td>NAME</td>
<td>The name of the segment cached in the Buffer Cache</td>
</tr>
<tr>
<td>KIND</td>
<td>The type of segment that is cached in the Buffer Cache (e.g., table, Index)</td>
</tr>
<tr>
<td>OWNER#</td>
<td>The identifier assigned to the database user who owns the segment whose blocks are cached in the Buffer Cache</td>
</tr>
</tbody>
</table>
Before using the V$CACHE view, you must create it by running the script supplied by Oracle called CATPARR.SQL. This script should be run while logged in as the user SYS. This script can be found in $ORACLE_HOME/rdbms/admin on Unix systems and %ORACLE_HOME%\rdbms\admin on Windows 2000 systems. The V$CACHE dynamic performance view, like V$BH, is actually intended for use in tuning Oracle9i Real Application Clusters.

By querying the V$CACHE and DBA_USERS views using the common columns OWNER# and USER_ID, you can also see which objects are currently cached in the Buffer Cache; how many Buffer Cache buffers those objects are using; and the owner, name, and type of those objects:

```
SQL> SELECT username "Owner",
          name 'Seg. Name',
          kind 'Seg. Type',
          COUNT(DISTINCT block#) 'Num. Buffers'
       FROM v$cache, dba_users
       WHERE v$cache.owner#=dba_users.user_id
       GROUP BY name, username, kind
       HAVING COUNT(DISTINCT block#) > 10
       ORDER BY 3 DESC;
```

<table>
<thead>
<tr>
<th>Owner</th>
<th>Seg. Name</th>
<th>Seg. Type</th>
<th>Num. Buffers</th>
</tr>
</thead>
<tbody>
<tr>
<td>APPS</td>
<td>EMPLOYEE</td>
<td>TABLE</td>
<td>12</td>
</tr>
<tr>
<td>APPS</td>
<td>EMPLOYEE_PK_IDX</td>
<td>INDEX</td>
<td>10</td>
</tr>
<tr>
<td>APPS</td>
<td>REGION</td>
<td>TABLE</td>
<td>8</td>
</tr>
</tbody>
</table>

Only segments having at least 10 blocks cached in the Buffer Cache are displayed by this query. A more (or less) restrictive HAVING clause can be used to make the query more (or less) selective.
Using the output from this query we again see that the EMPLOYEE and REGION tables along with the EMPLOYEE_PK_ID index are currently cached in memory so they may make good candidates for the Keep or Recycle Pool. For the purposes of our example, assume you have determined that the application segments shown in Table 5.10, along with several others, should be specifically cached in the pools shown. Any other application segments not specifically allocated to the Keep or Recycle Pools will use the Default Pool.

<table>
<thead>
<tr>
<th>Segment Name</th>
<th>Segment Type</th>
<th>Buffer Pool</th>
</tr>
</thead>
<tbody>
<tr>
<td>EMPLOYEE</td>
<td>Table</td>
<td>Keep Pool</td>
</tr>
<tr>
<td>REGION</td>
<td>Table</td>
<td>Keep Pool</td>
</tr>
<tr>
<td>DIVISION</td>
<td>Table</td>
<td>Recycle Pool</td>
</tr>
<tr>
<td>EMPLOYEE_FIRST_NAME_IDX</td>
<td>Index</td>
<td>Keep Pool</td>
</tr>
<tr>
<td>EMPLOYEE_ID_PK</td>
<td>Index</td>
<td>Keep Pool</td>
</tr>
<tr>
<td>SALES_HISTORY</td>
<td>Table</td>
<td>Recycle Pool</td>
</tr>
</tbody>
</table>

Oracle recommends you consider caching in the Keep Pool frequently accessed segments whose total size is less than 10 percent of the Default Pool size.

Oracle recommends that you consider caching in the Recycle Pool segments whose blocks will not be accessed outside of individual transactions, and segments whose total size is more than twice the size of the Default Pool.

Before you can actually cache these segments in their appropriate Buffer Pools, you will need to determine how large the Keep and Recycle Pools will need to be in order to hold each segment’s data.

Oracle Objective: Create and size multiple Buffer Pools
Determining the Size of Each Pool

After determining which segments you wish to cache, you need to determine how large each pool should be. Making this determination is easy if the segments to be cached have been analyzed. The BLOCKS column of the DBA_TABLES and DBA_INDEXES data dictionary views can be used to determine how many Database Buffer Cache buffers would be needed to cache an entire segment or portion of that segment. For the purposes of our example, assume you have decided to create three Buffer Pools with the following sizes in order to cache the six segments shown in Table 5.10:

- Keep Pool: 150MB
- Recycle Pool: 50MB
- Default Pool: 300MB

Figure 5.11 shows what these three Buffer Pools would conceptually look like in the SGA once they are created.

Creating the Buffer Pools

Next, the Buffer Pools must be created with the selected sizes. The Buffer Pools can either be created dynamically using the ALTER SYSTEM command, or manually by changing init.ora and then shutting down and restarting the instance.

Dynamically Creating Buffer Pools

If the value assigned to SGA_MAX_SIZE allows it, you can dynamically create the Keep and Recycle Pools using the ALTER SYSTEM command:

```sql
SQL> ALTER SYSTEM SET db_cache_size=300M;
```
Chapter 5: Tuning the Database Buffer Cache

SQL> ALTER SYSTEM SET db_keep_cache_size=150M;
SQL> ALTER SYSTEM SET db_recycle_cache_size=50M;

If the value for \texttt{SGA\_MAX\_SIZE} is set too low to allow you to dynamically add the two additional Buffer Pools, then you will have to set them manually.

**Manually Creating Buffer Pools**

The \texttt{SGA\_MAX\_SIZE} parameter must be set to a suitably large value to accommodate all the SGA components. If the value is too low, you may not be able to create the Keep and Recycle Pools. You can edit \texttt{init.ora}, and increase the values for \texttt{SGA\_MAX\_SIZE}, \texttt{DB\_CACHE\_SIZE}, \texttt{DB\_KEEP\_CACHE\_SIZE}, and \texttt{DB\_RECYCLE\_CACHE\_SIZE} all at one time, before shutting down and restarting the instance for them to take effect. These parameters were described previously in Table 5.4.

Using our desired values of 150MB for the Keep Pool, 50MB for the Recycle Pool, and 300MB for the Default Pool, we might configure our \texttt{init.ora} like this:

```sql
# Maximum size of the SGA, must be large enough to hold
# the Shared Pool, Buffer Cache, and Redo Log Buffer
SGA_MAX_SIZE = 1G
#
# Desired total size of the Default Buffer Pool
DB_CACHE_SIZE = 300M
#
# Desired total size of the Keep Buffer Pool
DB_KEEP_CACHE_SIZE = 150M
#
# Desired total size of the Recycle Buffer Pool
DB_RECYCLE_CACHE_SIZE = 50M
```

The Oracle8i \texttt{init.ora} parameters formerly used to create multiple Buffer Pools, \texttt{DB\_BLOCK\_BUFFERS}, \texttt{BUFFER\_POOL\_KEEP}, \texttt{BUFFER\_POOL\_RECYCLE}, and \texttt{DB\_BLOCK\_LRU\_LATCHES}, should no longer be used in Oracle9i as they will be made obsolete in future Oracle releases. Use the new \texttt{init.ora} parameters discussed here instead.

Once the Buffer Pools are created with the appropriate sizes, you can assign the segments to the appropriate pools.
Assigning Segments to Pools

Now that the appropriate Buffer Pools are created, you must use the ALTER command to assign each segment to its specified Buffer Pool. The following example assigns each of the segments we identified in the previous section to their appropriate Buffer Pools:

```
SQL> ALTER TABLE apps.employee
  2  STORAGE (BUFFER_POOL KEEP);

SQL> ALTER TABLE apps.region
  2  STORAGE (BUFFER_POOL KEEP);

SQL> ALTER TABLE apps.division
  2  STORAGE (BUFFER_POOL RECYCLE);

SQL> ALTER INDEX apps.employee_first_name_idx
  2  STORAGE (BUFFER_POOL KEEP);

SQL> ALTER INDEX apps.employee_id_pk
  2  STORAGE (BUFFER_POOL KEEP);

SQL> ALTER TABLE apps.sales_history
  2  STORAGE (BUFFER_POOL RECYCLE);
```

You can view which segments have been assigned non-Default Buffer Pools by using the following query on `DBA_SEGMENTS`:

```
SQL> SELECT owner, segment_type, segment_name, buffer_pool
  FROM dba_segments
WHERE buffer_pool != 'DEFAULT';
```

<table>
<thead>
<tr>
<th>OWNER</th>
<th>SEGMENT_TYPE</th>
<th>SEGMENT_NAME</th>
<th>BUFFER_</th>
</tr>
</thead>
<tbody>
<tr>
<td>APPS</td>
<td>TABLE</td>
<td>EMPLOYEE</td>
<td>KEEP</td>
</tr>
<tr>
<td>APPS</td>
<td>TABLE</td>
<td>REGION</td>
<td>KEEP</td>
</tr>
<tr>
<td>APPS</td>
<td>TABLE</td>
<td>DIVISION</td>
<td>RECYCLE</td>
</tr>
<tr>
<td>APPS</td>
<td>INDEX</td>
<td>EMPLOYEE_FIRST_NAME_IDX</td>
<td>KEEP</td>
</tr>
<tr>
<td>APPS</td>
<td>INDEX</td>
<td>EMPLOYEE_ID_PK</td>
<td>KEEP</td>
</tr>
<tr>
<td>APPS</td>
<td>TABLE</td>
<td>SALES_HISTORY</td>
<td>RECYCLE</td>
</tr>
</tbody>
</table>
Any segment buffers that were cached prior to the ALTER command will remain in their current pool. The buffers will not be moved to the newly specified Buffer Pool until the next time they are re-read from disk.

**Monitoring the Performance of Multiple Buffer Pools**

After making the appropriate changes to the init.ora and specifying which segments to cache in which Buffer Pools, you can monitor the performance of the Buffer Pools using the V$BUFFER_POOL and V$BUFFER_POOL_STATISTICS dynamic performance views.

**Oracle Objective**

Monitor the use of the Buffer Cache, also the different pools within the Buffer Cache

**Using V$BUFFER_POOL**

The V$BUFFER_POOL dynamic performance view contains information about the configuration of the multiple Buffer Pools themselves. Table 5.11 describes the columns that are useful for obtaining information about the individual Buffer Pools.

<table>
<thead>
<tr>
<th>Table 5.11</th>
<th>Partial Description of V$BUFFER_POOL Columns</th>
</tr>
</thead>
<tbody>
<tr>
<td>Column</td>
<td>Description</td>
</tr>
<tr>
<td>NAME</td>
<td>The name of the Buffer Pool</td>
</tr>
<tr>
<td>BLOCK_SIZE</td>
<td>The block size for the buffers in this Buffer Pool, expressed in bytes</td>
</tr>
<tr>
<td>CURRENT_SIZE</td>
<td>The current size of the Buffer Pool, expressed in megabytes</td>
</tr>
</tbody>
</table>

You can use these columns to query this view and examine the size of each individual Buffer Pool:

```
SQL> SELECT name, block_size, current_size
     2   FROM v$buffer_pool;
```
Using `V$BUFFER_POOL_STATISTICS`  
The `V$BUFFER_POOL_STATISTICS` dynamic performance view contains several columns that contain information that is useful for calculating hit ratios for each of the individual Buffer Pools. Table 5.12 describes the columns that are used in calculating the hit ratios.

### TABLE 5.12 Partial Description of `V$BUFFER_POOL_STATISTICS` Columns

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAME</td>
<td>The name of the Buffer Pool</td>
</tr>
<tr>
<td>DB_BLOCK_GETS</td>
<td>Number of requests for segment data that were satisfied by using cached segment buffers</td>
</tr>
<tr>
<td>CONSISTENT_GETS</td>
<td>Number of requests for segment data that were satisfied by using cached rollback blocks for read consistency</td>
</tr>
<tr>
<td>PHYSICAL_READS</td>
<td>The number of requests for segment data that required that the data be read from disk into the Buffer Cache</td>
</tr>
</tbody>
</table>

You can use the statistics in `V$BUFFER_POOL_STATISTICS` to calculate hit ratios for each of the individual Buffer Pools using the following query:

```sql
SQL> SELECT name 'Buffer Pool',
       1- (physical_reads / (db_block_gets + consistent_gets)) 'Buffer Pool Hit Ratio'
       2 FROM v$buffer_pool_statistics
       3 ORDER BY name;
```
As you might expect, if you have tuned the multiple Buffer Pools properly, the hit ratio should be very high for the Keep Pool and low for the Recycle Pool. The hit ratio for the Default Pool should be somewhere in between these two values, but typically should be in the range of 70–80 percent.

<table>
<thead>
<tr>
<th>Buffer Pool</th>
<th>Buffer Pool Hit Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEFAULT</td>
<td>.713179916</td>
</tr>
<tr>
<td>KEEP</td>
<td>.9721</td>
</tr>
<tr>
<td>RECYCLE</td>
<td>.238907</td>
</tr>
</tbody>
</table>

**Real World Scenario**

**100 Percent Hit Ratios Don’t Always Mean Top Performance**

Your users have been complaining about poor application performance, but your queries on V$SYSSTAT show a Buffer Cache hit ratio of 99.97 percent. Since this hit ratio indicates little room for improvement, you consider moving on to other areas in search of possible tuning improvements.

In this situation, close inspection should be made of the contents of the Buffer Cache. In particular, you should determine how many index buffers are cached in the Buffer Cache and to which indexes they belong. Assume that, using V$CACHE and DBA_USERS, you determine that the index on the ORDER_STATUS column of the very large ORDERS table is using many of the Buffer Cache’s buffers. Upon further inspection, you also determine that there are only three possible values for ORDER_STATUS: Open, Closed, and Hold. Additionally, you discover that 90 percent of the records in the ORDERS table have a status of Closed.

The unselective index on the ORDER_STATUS column of the ORDERS table is a likely suspect for your application’s poor performance, yet high Buffer Cache hit ratio. As indexes of this type are scanned, many of their blocks are brought into the Buffer Cache. If the index is on a “popular” application table—like ORDERS—then the Buffer Cache hit ratio becomes artificially inflated because of the large number of index blocks cached in the Buffer Cache and the frequency with which they are requested by the application. This causes a high hit ratio, but low overall application performance.
Several steps can be taken to fix this situation. The first would be to make sure that current statistics are kept in the data dictionary so that the cost-based optimizer can determine the selectivity of the index and forego its use when a full table scan is more effective. A second way to minimize this problem is to create histograms on the ORDERS table so that the cost-based optimizer gets a better picture of the distribution of values within the ORDER_STATUS column. Finally you may want to designate that the buffers from the index be cached in the Recycle Pool instead of the Buffer Cache. All of these steps will not only help maintain a high Buffer Cache hit ratio, but also make sure that it provides real performance benefits for your application users.

**Cache Tables in Memory**

No matter how many Buffer Pools you decide to use, each one is still managed by an LRU List. Normally, blocks that are accessed by application users are placed at the most recently used end of the LRU List. However, as was noted earlier in this chapter, tables that are accessed via a full table scan (FTS) place their blocks immediately at the least recently used end of the LRU List.

This behavior can present an interesting tuning dilemma. If the cost-based optimizer sees that a frequently used table (like a validation or lookup table) is small, it is likely to be accessed with an FTS. But this FTS places the table’s buffers at the least recently used end of the LRU List where they are quickly removed from the Buffer Cache, only to be re-read into memory when they are needed again by a subsequent application user.

One way to manage this problem, particularly with small tables, is to make use of cached tables. Tables designated as being *cache tables* do not have their buffers placed at the least recently used end of the LRU List when they are accessed via an FTS. Instead, these buffers are placed at the most recently used end of the LRU List just as if they had not been full table scanned. This has the effect of keeping these buffers in memory longer while still accessing them in the most efficient manner.Cached tables can be implemented in three ways: at table creation, by altering the table after creation, or by using a hint.
Caching at Table Creation

You can make a table a cache table by including the keyword CACHE when the table is created, as follows:

```
SQL> CREATE TABLE phone_list
2  ( employee_id   number,
3  phone_number  varchar2(11),
4  extension     varchar2(4))
5  TABLESPACE appl_tab
6  STORAGE (INITIAL 50K NEXT 50K PCTINCREASE 0)
7  CACHE;
```

By default, all tables are created as NOCACHE tables unless the CACHE option is specified.

Using ALTER TABLE to Create Cache Table

You can change an existing table into a cache table using the keyword CACHE with the ALTER TABLE command:

```
SQL> ALTER TABLE employee CACHE;
```

The NOCACHE option is used to change a table back to its regular FTS behavior.

Using Hints to Cache

You can also dynamically CACHE or NOCACHE tables using the appropriate hint:

```
SQL> SELECT /*+ CACHE */ last_name, first_name
2  FROM employee;
```

The hint will only affect this query; all other accesses of the EMPLOYEE table would still use the table's default cache mode.

Displaying Cache Table Information

You can use the CACHE column in the DBA_TABLES view to determine which tables are cached, as shown here:

```
SQL> SELECT owner, table_name
2  FROM dba_tables
3  WHERE LTRIM(cache) = 'Y';
```
**Bypass Buffer Cache**

Another technique for improving the performance of the Database Buffer Cache is to bypass the Buffer Cache completely for certain types of buffer requests. By bypassing the Buffer Cache, buffers that are already stored there are not moved down or off the LRU List during processing. These two types of transactions can be configured to bypass the Database Buffer Cache:

- Sort Direct Writes (Covered in Chapter 8, “Tuning Disk I/O”)
- Parallel DML

Parallel Data Manipulation Language (DML) involves doing bulk inserts, updates, and deletes in parallel by starting multiple processes to perform the action. Several *init.ora* parameters must be set properly before parallel execution can be used effectively.

**Use Indexes Appropriately**

Another big impact you can make on the performance of the Database Buffer Cache—from an application SQL code perspective—is to ensure that unnecessary full table scans are avoided through the appropriate use of indexes. The fewer full table scans that occur, the higher the Database Buffer Cache hit ratio will be. One simple way to try to encourage the use of indexes is to build indexes on the foreign key columns of tables that reference a primary
key column in another table. This will not only help to improve the effectiveness of multi-table joins, but it will also minimize Buffer Cache intensive FTS sort-and-merge joins between the two tables.

**Summary**

The role of the SGA’s Database Buffer Cache is to cache the most frequently used segments so application users can minimize physical disk I/O when accessing data. The size of the Database Buffer Cache is specified by the init.ora parameters SGA_MAX_SIZE, DB_CACHE_SIZE, DB_KEEP_CACHE_SIZE, and DB_RECYCLE_CACHE_SIZE. Each Buffer Cache buffer holds the data from one segment block. The activity of the Database Buffer Cache is managed by an LRU algorithm, a Dirty List, user Server Processes, and Database Writer (DBW0).

The performance of the Database Buffer Cache is measured by a hit ratio that compares the total number of reads performed in the instance to the number of reads that were done from disk. A Buffer Cache hit ratio of 90 percent or better is expected for well-tuned OLTP systems. The hit ratio can be calculated by querying V$SYSSTAT and the output from STATSPACK and REPORT.TXT. Other indicators of Buffer Cache performance include Free Buffer Inspected, Buffer Busy Wait, and Free Buffer Wait statistics and Graphical representations Buffer Cache performance using the Oracle Enterprise Manager Performance Manager tool.

The options for improving the hit ratio of the Database Buffer Cache include: making the Buffer Cache larger, using multiple Buffer Pools, caching tables in the Buffer Cache, bypassing the Buffer cache for I/O intensive operations, and using indexes to minimize full table scans appropriately.

**Exam Essentials**

Know how the Buffer Cache is managed and utilized. Be able to describe how data is moved into and out of the Buffer Cache and the LRU mechanisms that manage the buffers located there.
Understand how to measure the performance of the Buffer Cache. Know how to determine the Buffer Cache hit ratio using output from dynamic performance views, STATSPACK, and REPORT.TXT. Be able to describe what secondary statistics can be used as an indication of Buffer Cache performance.

Understand how to create and utilize multiple Buffer Pools. Be able to create the three types of Buffer Pools dynamically and manually. Know how to assign segments to the individual Buffer Pools and then monitor their performance.

Describe the benefits of table caching. Know how to cache tables in the Buffer Cache and what types of tables make good candidates for caching.

Key Terms

Before you take the exam, be certain you are familiar with the following terms:

- Buffer Cache Advisory hit ratio
- Buffer Pool Keep Pool
- cache table LRU list
- Database Buffer Cache Recycle Pool
- Default Pool user Server Process
- Dirty List Write List
Review Questions

1. Which of the following mechanisms does the Database Buffer Cache use to manage its memory?
   A. LRU list
   B. Dirty List
   C. DBW0
   D. All of the above play a role in managing the Database Buffer Cache.

2. Which of the following is not cached in the Database Buffer Cache?
   A. Tables
   B. Redo logs
   C. Indexes
   D. Rollback segments

3. Which of the following processes copy the segment blocks from disk into the Database Buffer Cache buffers?
   A. DBW0
   B. LGWR
   C. Server Process
   D. PMON

4. Unless a full table scan occurs, at which end of the LRU list does a user’s Server Process place its buffers in the Database Buffer Cache?
   A. MRU
   B. LRU
   C. Middle
   D. None of the above
5. Each time a buffer on the Database Buffer Cache LRU List is accessed, it is moved to which part of the LRU list?
   A. MRU
   B. LRU
   C. Middle
   D. None of the above

6. What is a buffer whose copy in the Database Buffer Cache does not match the copy of the segment block on disk called?
   A. Pinned buffer
   B. Free buffer
   C. Dirty buffer
   D. Rollback buffer

7. Which mechanism does a user’s Server Process use when locating a free buffer in the Database Buffer Cache?
   A. LRU List
   B. Dirty List
   C. DBW0
   D. All of the above

8. At which of the following times does DBW0 write dirty buffers to disk?
   A. When the Dirty List reaches a threshold length
   B. When a user’s Server Process searches the LRU List too long without finding a free buffer
   C. At a checkpoint
   D. All of the above

9. Choose the primary measure of the performance of the Database Buffer Cache:
A. Average LRU list length
B. Maximum Dirty List miss ratio
C. Database Buffer Cache hit ratio
D. All of the above

10. Which of the following dynamic performance views is used to calculate the Database Buffer Cache hit ratio?
   A. V$BUFFERCACHE
   B. V$SYSTEMSTATS
   C. V$SYSSTAT
   D. V$SGASTAT

11. According to Oracle, what hit ratio should a well-tuned Database Buffer Cache in an OLTP system have?
   A. 10 percent or less
   B. 50 percent
   C. 60–80 percent
   D. 90 percent or higher

12. Which of the following Database Buffer Cache pools is/are required?
   A. Keep Pool
   B. Default Pool
   C. Recycle Pool
   D. All of the above are required.

13. When a user issues a COMMIT statement, which Oracle process eventually writes the information from the Database Buffer Cache onto disk?
   A. SMON
   B. DBW0
   C. PMON
   D. LGWR
14. Given the following query output from V$DB_CACHE_ADVICE, what would be the optimal size of the database buffer cache without wasting resources?

<table>
<thead>
<tr>
<th>NAME</th>
<th>SIZE_FOR_ESTIMATE</th>
<th>ESTD_PHYSICAL_READS</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEFAULT</td>
<td>10.1797</td>
<td>7686</td>
</tr>
<tr>
<td>DEFAULT</td>
<td>20.3594</td>
<td>6614</td>
</tr>
<tr>
<td>DEFAULT</td>
<td>30.5391</td>
<td>6028</td>
</tr>
<tr>
<td>DEFAULT</td>
<td>40.7188</td>
<td>5785</td>
</tr>
<tr>
<td>DEFAULT</td>
<td>50.8984</td>
<td>5497</td>
</tr>
<tr>
<td>DEFAULT</td>
<td>61.0781</td>
<td>5342</td>
</tr>
<tr>
<td>DEFAULT</td>
<td>71.2578</td>
<td>4915</td>
</tr>
<tr>
<td>DEFAULT</td>
<td>81.4375</td>
<td>4867</td>
</tr>
<tr>
<td>DEFAULT</td>
<td>91.6172</td>
<td>4866</td>
</tr>
<tr>
<td>DEFAULT</td>
<td>101.7969</td>
<td>4866</td>
</tr>
<tr>
<td>DEFAULT</td>
<td>111.9766</td>
<td>4866</td>
</tr>
<tr>
<td>DEFAULT</td>
<td>122.1563</td>
<td>4866</td>
</tr>
<tr>
<td>DEFAULT</td>
<td>132.3359</td>
<td>4866</td>
</tr>
</tbody>
</table>

A. 70MB  
B. 50MB  
C. 100MB  
D. 132MB

15. Which of these parameters does not have an effect on the size of the Database Buffer Cache?

A. DB_KEEP_CACHE_SIZE  
B. DB_BLOCK_SIZE  
C. DB_MAX_CACHE_SIZE  
D. DB_RECYCLE_CACHE_SIZE

16. Why does increasing the size of the Database Buffer Cache improve performance?
A. Server Processes are more likely to find free buffers when needed.

B. Buffers are less likely to be moved out of the cache prematurely by the LRU mechanism.

C. Both of the above are correct.

D. Neither of the above is correct.

17. Which of the following Buffer Pools can be used to cache segments you have determined to be frequently accessed?

A. Kept

B. Keep

C. Cache

D. Pin

18. Which of the following views can be used to determine which segments might make good candidates for the Keep Pool?

A. V$BH

B. DBA_OBJECTS

C. V$CACHE

D. Any of the above

19. Choose the three component pool types available in the database buffer cache.

A. Default

B. Common

C. Keep

D. Recycle

20. What are tables called whose buffers are placed at the most recently used end of the LRU List, even when full table scanned?

A. Pinned tables

B. Cache tables

C. Kept tables

D. None of the above
Answers to Review Questions

1. D. The LRU list is used to keep frequently accessed buffers in memory. The Dirty List keeps track of buffers whose copy in memory has changed. Database Writer (DBW0) writes the dirty buffers to disk.

2. B. Table, index, cluster, LOB indexes, LOB segments, rollback segments, and temporary segments can all be cached in the Database Buffer Cache.

3. C. Each user’s Server Process copies the requested blocks into the Database Buffer Cache.

4. A. Segment blocks are copied into buffers that are placed at the most recently used end of the LRU List.

5. A. Once accessed, any cached buffers are moved from their current location on the LRU List to the beginning, or most recently used (MRU) end, of the LRU list.

6. C. Dirty buffers store data that has been changed by the application user but has not yet been written to disk by DBW0.

7. D. As the Server Process searches the LRU List for free buffers, dirty buffers are moved to the Dirty List where they are written to disk by DBW0 when the Dirty List reaches its threshold length.

8. D. In addition to these times, DBW0 also writes dirty buffers to disk when tablespaces that store blocks for those segments are placed in hot backup or offline mode and when segments are dropped.

9. C. The Database Buffer Cache hit ratio finds the percentage of buffer requests able to be satisfied using the existing cached buffers.

10. C. The V$SYSSTAT dynamic performance view contains statistics on the total number of buffers read and the number of those that were read from disk.

11. D. Hit ratios may be lower for non-OLTP systems.

12. B. Only the Default Pool is required. The Keep and Recycle Pools can be configured to enhance Buffer Cache performance after frequently and infrequently accessed segments have been identified, respectively.

13. B. Database Writer is responsible for writing blocks from the Database Buffer Cache to the datafiles on disk.
14. A. The best setting for the `DB_CACHE_SIZE` would be 70MB because any increases beyond this point yield only a small reduction of physical reads.

15. C. `DB_MAX_CACHE_SIZE` is not a valid initialization parameter.

16. C. Increasing the size of the Database Buffer Cache makes more free buffers available and cuts down on unwanted LRU flushing of buffers.

17. B. You can assign frequently accessed application segments in the Keep Buffer Pool by using the `STORAGE (BUFFER_POOL KEEP)` parameter with the `ALTER` command.

18. D. The `V$BH` and `V$CACHE` dynamic performance views can be used to determine which segments currently have their blocks cached in memory. Additionally, the `DBA_OBJECTS` data dictionary view can be joined to `V$BH` to determine the owner and name of each cached segment.

19. A, C, D. The Default, Keep, and Recycle pools are available in the Database Buffer Cache. You can specify which pool you want to place a segment into by utilizing the `BUFFER_POOL` syntax on the storage clause for the table or index segment when defining or altering the segment definition.

20. B. Tables that have been created or altered using the `CACHE` keyword, or accessed using the `CACHE` hint, will have their buffers placed at the most recently used end of the LRU list during a full table scan.
Chapter 6

Tuning Other SGA Areas

ORACLE9i PERFORMANCE TUNING EXAM
OBJECTIVES COVERED IN THIS CHAPTER:

✓ Describe the UGA and session memory considerations
✓ Set the Large Pool
✓ Identify issues associated with managing users in a Shared Server environment
✓ Diagnose and resolve performance issues with Oracle Shared Server processes
✓ Configure the Oracle Shared Server environment to optimize performance
✓ Monitor and size the Java Pool
✓ Control the amount of Java session memory used by a session

Exam objectives are subject to change at any time without prior notice and at Oracle’s sole discretion. Please visit Oracle’s Certification website (http://education.oracle.com/certification/) for the current exam objectives listing.
Thus far, you have examined tuning techniques for two SGA components: the Shared Pool and the Database Buffer Cache. These two SGA components, and the Redo Log Buffer discussed in Chapter 7, “Tuning Redo Mechanisms”, are used by every Oracle application. This chapter, however, will focus on tuning three areas within the SGA that may or may not be utilized in all environments:

- Shared Pool architectural components
- Large Pool
- Java Pool

Because some Oracle environments do not make use of these SGA components, this chapter will discuss the essential concepts related to each of these areas before introducing the techniques used to monitor and enhance their performance.

Shared Server Concepts

Each Oracle application user has two processes associated with their connection to the database: a User Process, which executes on the user’s client machine or on the application’s web server, and a Server Process, which executes on the Oracle Server machine.

In the default Oracle9i configuration, called Dedicated Server, each user’s Server Process is dedicated to servicing requests for that user’s session only. On Unix systems these Server Processes execute on the Oracle server as individual OS processes. If there are 50 connected users, then there will be 50 corresponding Server Processes running on the Oracle server machine to service their database activities.

A second optional configuration, called Shared Server, is also available. In this configuration, several Shared Server processes are created on the Oracle
server machine. These Shared Server processes can be used to send application SQL statements to, or return data from, the instance by any User Process that needs them.

In versions prior to Oracle9i, the Shared Server option was referred to as the Multithreaded Server (MTS) option.

You can think of the difference between Dedicated Servers and Shared Servers as being much like the difference between owning your own automobile and using taxicabs when traveling from place to place. If you own your own vehicle, you can go wherever you want to go, whenever you want to go there, without having to wait for access to an automobile. However, when you are not using your vehicle, when you’re asleep for example, that expensive resource goes completely unused. Conversely, if you use taxicabs as your mode of transportation, you may or may not have access to the vehicle precisely when you want it, but you will eventually get to where you want to go. However, other patrons can continuously utilize the taxicab during the times that you do not need it, making good use of that resource.

These distinctions are also true of Dedicated and Shared Servers. The Dedicated Server configuration provides for efficient communication between the User Process and the Server Process, because the user’s Server Process is dedicated to sending application SQL statements to, or returning data from, the instance for only that single user. However, just as with automobiles, Dedicated Servers have the disadvantage of consuming memory and CPU resources on the Oracle server machine—even when the user is idle. This can lead to scalability problems when the number of application users is large because the server resources required to support the user Server Processes are often not utilized efficiently.

The Shared Server configuration solves the scalability concerns of the Dedicated Server configuration by better utilizing the Oracle server machine’s available memory and CPU to support the activities of the application users. But, just as with taxicabs, Shared Servers can also negatively impact
performance if they are not available when User Processes need them. The
Shared Servers themselves are only one component of the Shared Server
architecture.

On Windows 2000 systems all user Server Processes execute within the OS
threads associated with the oracle.exe executable, not as discrete OS pro-
cesses as they do on Unix systems. Therefore, the Shared Server feature does
not provide the same scalability improvements on Windows 2000 systems that
it does on Unix systems.

Shared Server Architecture

When you need a taxicab, you have to first telephone the taxicab company
and notify them that you need a ride. The taxicab company will pass your
request to the taxicab dispatcher, who places your name on the list of per-
sons wanting a cab. The taxicab dispatcher then assigns your request to a
particular cab, which will eventually come to your address and pick you up,
and take you to your destination. The Shared Server architecture has several
components that perform similar functions. The components of the Shared
Server architecture are:

- User Processes
- Oracle Net Listener Process
- Dispatcher Processes
- Request Queue
- Shared Server processes
- Response Queue

Each of these components is discussed in the following sections.

User Process Each application user, whether they initiate their database
connection via a client-based executable or via a middle tier application
server, creates a User Process that manages their connection to the
Oracle server machine.

Oracle Net Listener The Oracle Net Listener listens for incoming client
requests for data or DML activity. When a request is detected, the listener
assigns the request to the least busy Dispatcher Process.
Dispatcher Process  Dispatchers are Oracle background processes that run on the Oracle server machine and accept incoming requests for database processing from the User Processes. Up to five dispatchers can be running at one time. Each Dispatcher process has a unique name. In a Unix environment these processes can be seen from the operating system using the `ps` command:

```
$ ps -ef |grep ora_d
oracle   13323     1  0 16:38 ?       00:00:00 ora_dbw0_PROD
oracle   13343     1  0 16:38 ?       00:00:00 ora_d000_PROD
oracle   13345     1  0 16:38 ?       00:00:00 ora_d001_PROD
oracle   13347     1  0 16:38 ?       00:00:00 ora_d002_PROD
oracle   13356 13257  0 16:39 pts/0    00:00:00 grep ora_d
```

The above output from the `ps` command shows that three Dispatcher processes, `ora_d000_PROD`, `ora_d001_PROD`, and `ora_d002_PROD`, are running against an instance called `PROD`. The Dispatcher Processes are spawned at instance startup and run continuously in memory even if no users are currently connected to the instance.

**WARNING** Killing a Dispatcher process using the Unix `kill` command will not only kill the Dispatcher process, but will also disconnect every user connection that is using that Dispatcher process to connect to the database.

Request Queue  The queue into which the Dispatcher places requests received from the User Processes is called the Request Queue. There is only one request queue. This queue is stored within the Oracle SGA.

Shared Server Process  Shared Server processes are Oracle background processes that run on the Oracle server machine, interacting with the SGA on behalf of the individual User Processes. The Shared Servers perform the same activities that a Dedicated Server process would (e.g., parse SQL statements, read segment blocks from disk into the Buffer Cache, etc.). There can be many Shared Server processes. The maximum number is OS specific. Each Shared Server process has a unique name. In a Unix environment these processes can be seen from the operating system using the `ps` command:

```
$ ps -ef |grep ora_s
oracle   13329     1  0 16:38 ?       00:00:00 ora_smon_PROD
```

The above output from the `ps` command shows that five Shared Server processes, `ora_s000_PROD`, `ora_s001_PROD`, `ora_s002_PROD`, `ora_s003_PROD`, `ora_s004_PROD`, and `ora_s005_PROD` are running against an instance called PROD. The Shared Server processes are spawned at instance startup and run continuously in memory even if no users are currently connected to the instance.

**Response Queue** The queue into which the Shared Server processes place the results of their database activity is called the **Response Queue**. The Dispatcher processes then return these results to the User Process that initiated the request. There is one response queue for each dispatcher. These queues are stored within the Oracle SGA.

Figure 6.1 shows how these components interact in the Shared Server configuration.

**FIGURE 6.1** Shared Server architecture
In order to effectively implement Shared Servers without compromising overall performance, several conditions must be met before a Shared Server configuration should be considered:

**Many Application Users** Implementing the Shared Server option generally only makes sense when the number of concurrent application users exceeds 200 or more users, depending on the Oracle server’s memory and CPU resources.

**Short Application Transactions** Since application users will be sharing Server Processes when communicating with the database, it is important that the application SQL perform short transactions. Applications that issue long running transactions would cause an individual user to monopolize a Shared Server process, causing other users wait for that resource.

**Noncontinuous Transactions** The application users should be utilizing the application in such a manner that occasional pauses occur in their activities. These pauses will allow the Shared Server process to service other users’ requests more efficiently. Pauses of this type are common in OLTP environments such as order entry systems. Users of the order entry application generally pause frequently during transaction processing as they discuss the details of the order with the customer placing the order.

If your application meets the guidelines listed above, you will need to perform the actions described in the next section in order to configure the Shared Server feature.

**Configuring Shared Servers**

Several `init.ora` parameters must be properly configured before the Shared Server option can be implemented. These parameters are used to determine the number of Dispatcher and Shared Server processes that should be spawned at instance startup, and how they are managed once users begin accessing the database.
Configure the Oracle Shared Server environment to optimize performance

The `init.ora` parameters used to configure the Shared Server option are described in Table 6.1.

**TABLE 6.1** Shared Server `init.ora` Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DISPATCHERS</td>
<td>Used to specify the number of Dispatchers to spawn at instance startup and the Oracle Net protocol assigned to each Dispatcher. Valid values are 0 to 5.</td>
</tr>
<tr>
<td>MAX_DISPATCHERS</td>
<td>Used to specify the maximum number of Dispatcher processes allowed against the instance. The default value is 5; the maximum value is 5 or the value for DISPATCHERS, whichever is greater.</td>
</tr>
<tr>
<td>SHARED_SERVERS</td>
<td>Used to specify the number of Shared Server processes to spawn at instance startup. When set to a value of 0, the Shared Server option is disabled. The minimum value for the Shared Server option is 1; the maximum value is OS dependent.</td>
</tr>
<tr>
<td>MAX_SHARED_SERVERS</td>
<td>Used to specify the maximum number of Shared Server processes allowed against the instance. The default value is either 20 or $2 \times$ value of SHARED_SERVERS; the maximum is OS dependent.</td>
</tr>
<tr>
<td>CIRCUITS</td>
<td>Used to specify the number of virtual circuits for handling incoming and outgoing network connections.</td>
</tr>
<tr>
<td>PROCESSES</td>
<td>Used to specify the maximum number of OS processes that can access the instance. The default value is OS dependent, but must often be increased when using the Shared Server option to allow for the additional processes.</td>
</tr>
</tbody>
</table>
The init.ora parameters shown in Table 6.1 replace the deprecated Oracle8i parameters MTS_DISPATCHERS, MTS_MAX_DISPATCHERS, MTS_SERVERS, MTS_MAX_SERVERS, and MTS_CIRCUITS.

While the DISPATCHERS and SHARED_SERVERS parameters are used to establish the minimum number of those processes to spawn at instance startup, both components can be adjusted upward to the ceiling values specified by MAX_DISPATCHERS and MAX_SHARED_SERVERS. However, while Dispatcher processes must be explicitly started by the DBA using the ALTER SYSTEM command, Shared Server processes can be dynamically started and stopped by the PMON background process based on the demands placed on those resources by the connected application users.

Determining how many Dispatchers and Shared Servers to spawn at instance startup can be determined by monitoring the performance of the Shared Server environment once it has been initially configured.

Database administrators cannot use Shared Server connections to perform DBA operations like database startups, shutdowns, etc.

**Measuring Shared Server Performance**

Once configured the Shared Server option must be monitored and tuned to obtain optimal performance. Just as with the Shared Pool and Database Buffer Cache, several dynamic performance views can be used to monitor the performance of the Shared Server environment. These dynamic performance views are described in Table 6.2.

<table>
<thead>
<tr>
<th>View Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>V$SHARED_SERVER</td>
<td>Contains detailed statistics about the Shared Server processes, including idle and busy times for each Shared Server process.</td>
</tr>
<tr>
<td>V$QUEUE</td>
<td>Contains information about contention for Shared Server Request and Response queues.</td>
</tr>
</tbody>
</table>
Tuning Other SGA Areas

Each of the above views can be used to monitor the two dynamic components of the Shared Server architecture: Shared Server processes and Dispatcher Processes.

**Table 6.2** Shared Server Dynamic Performance Views (continued)

<table>
<thead>
<tr>
<th>View Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>V$SHARED_SERVER_MONITOR</td>
<td>Contains statistics regarding the Shared Server processes, including the Shared Servers that have been dynamically started and terminated by PMON since instance startup.</td>
</tr>
<tr>
<td>V$DISPATCHER</td>
<td>Contains detailed statistics about the Shared Server Dispatcher processes, including idle and busy times for each Dispatcher process.</td>
</tr>
<tr>
<td>V$DISPATCHER_RATE</td>
<td>Contains historical and real-time statistics regarding Dispatcher activity.</td>
</tr>
<tr>
<td>V$CIRCUIT</td>
<td>Contains statistics about the path, or circuit, a User Process has taken through a Dispatcher, Request Queue, Shared Server, Response Queue, and back to the User Process again.</td>
</tr>
</tbody>
</table>

Each of the above views can be used to monitor the two dynamic components of the Shared Server architecture: Shared Server processes and Dispatcher Processes.

**Oracle Objective**

Diagnose and resolve performance issues with Oracle Shared Server processes

**Monitoring Shared Server Process Performance**

One way to monitor the performance of the Shared Server processes is to monitor at what rate the available server processes are servicing database requests. This *busy ratio* can be calculated using the V$SHARED_SERVER dynamic performance view. Selected columns from this view that are useful for monitoring Shared Server busy ratios are shown in Table 6.3.

Figure 6.2 shows how, by querying these columns, a Shared Server busy ratio can be calculated.
The output from the above query shows that, at most, the Shared Server processes were busy servicing database requests 16.85 percent of the time.

High or increasing Shared Server busy ratios indicate that additional Shared Server processes may need to be spawned at instance startup.

When User Processes make a database request using a Shared Server, their requests are initially placed in the Request Queue by the Dispatcher process. If there are too few Shared Server processes, these requests may experience a delay while waiting to be serviced. These waits for access to a Shared Server process can be monitored using the V$QUEUE dynamic performance view. Selected columns from this view that are useful for monitoring contention for Shared Servers are shown in Table 6.4.
Chapter 6 • Tuning Other SGA Areas

Tuning Other SGA Areas

By querying these columns, an average wait time for access to a Shared Server can be calculated. This type of query is shown in Figure 6.3.

FIGURE 6.3 Calculating average wait times for Shared Servers

```
SQL> SELECT decode(totalq, 0, 'TOTALQ IS ZERO',
2 ROUND(sum(wait)/sum(totalq)), 4) "AVG DISPATCHER WAIT"
3 FROM v$queue
4 WHERE type = 'COMMON'
5 GROUP BY totalq;

AVG DISPATCHER WAIT
-------------
 0.0003
```

The output from the above query shows that, on average, User Processes waited .0003 hundredths of a second for a Shared Server process to service their database requests.

High or increasing waits for Shared Server processes may indicate a need to spawn additional Shared Server processes at instance startup.

Monitoring Dispatcher Performance

Like Shared Servers, the performance of the Dispatcher processes can also be monitored using a “busy ratio.” This ratio can be calculated using the V$DISPATCHER dynamic performance view. Selected columns from this view that are useful for monitoring Dispatcher busy ratios are shown in Table 6.5.

By querying these columns, a Dispatcher busy ratio can be calculated using the SQL statement shown in Figure 6.4.
**TABLE 6.5** Selected Columns from V$DISPATCHER

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NETWORK</td>
<td>Protocol used by the Dispatcher (network address)</td>
</tr>
<tr>
<td>STATUS</td>
<td>Current status of the Dispatcher. Possible Dispatcher statuses are: WAIT (idle) SEND (sending a message) RECEIVE (receiving a message) CONNECT (establishing connection) DISCONNECT (disconnecting a session) BREAK (performing a break) TERMINATE (terminating a connection) ACCEPT (accepting connections) REFUSE (refusing connection)</td>
</tr>
<tr>
<td>OWNED</td>
<td>Number of virtual circuits owned by this Dispatcher</td>
</tr>
<tr>
<td>BUSY</td>
<td>Total time (in 100ths of second) that the Dispatcher was busy</td>
</tr>
<tr>
<td>IDLE</td>
<td>Total time (in 100ths of second) that the Dispatcher was idle</td>
</tr>
</tbody>
</table>

**FIGURE 6.4** Query to determine Dispatcher busy ratio

```sql
SQL> SELECT name "DISPATCHER", network, (ROUND(SUM(busy)/(SUM(busy)+SUM(idle)),4)*100) "BUSY_RATE"
  2  FROM v$dispatcher
  3  GROUP BY name, network;
```

<table>
<thead>
<tr>
<th>DISPATCHER NETWORK</th>
<th>BUSY_RATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>D000 (ADDRESS=(PROTOCOL=tcp)(HOST=Moderne)(PORT=45280))</td>
<td>33.23</td>
</tr>
<tr>
<td>D001 (ADDRESS=(PROTOCOL=tcp)(HOST=Moderne)(PORT=45282))</td>
<td>27.12</td>
</tr>
<tr>
<td>D003 (ADDRESS=(PROTOCOL=tcp)(HOST=Moderne)(PORT=45283))</td>
<td>17.29</td>
</tr>
</tbody>
</table>

The output from the above query shows that, at most, the three Dispatcher processes for the TCP/IP network protocol were busy servicing database requests 33.23 percent of the time.

Oracle recommends adding additional Dispatcher processes if the Dispatcher busy rate consistently exceeds 50 percent.
Another indication of Shared Server performance is the amount of time that User Processes are waiting to have their requests accepted by the Dispatchers. The V$QUEUE and V$DISPATCHER dynamic performance views can be used to calculate the average wait time (in 100\textsuperscript{th} of second) experienced by User Processes waiting for access to Dispatcher processes using the query shown in Figure 6.5.

**FIGURE 6.5 Calculating Dispatcher Waits**

```
SQL> SELECT decode(totalq,0,'TOTALQ IS ZERO',
2          round(sum(wait)/sum(totalq)),4) "AVG DISPATCHER WAIT"
3     FROM v$queue q, v$dispatcher d
4     WHERE q.pqop='d',d.qid=q.pqid
5     AND q.type='dispatcher'
6     GROUP BY totalq;

AVG DISPATCHER WAIT
---------------------
          .0093
```

This output shows that, on average, User Processes waited .0093 hundredths of a second for a Dispatcher process to service their database requests.

High or increasing waits for Dispatcher processes may indicate a need to spawn additional Shared Server processes at instance startup.

The waits indicated by the above query on V$QUEUE and V$DISPATCHER are a sign of Dispatcher contention. Additional evidence that contention for Dispatchers is occurring in the Shared Server environment can be determined by querying the V$DISPATCHER_RATE dynamic performance view. Selected columns from this view that are useful for monitoring Dispatcher contention are shown in Table 6.6.

**TABLE 6.6 Selected Columns from V$DISPATCHER_RATE**

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAME</td>
<td>Name of the Dispatcher</td>
</tr>
<tr>
<td>CUR_IN_CONNECT_RATE</td>
<td>Current rate at which the Dispatcher is servicing requests from application users</td>
</tr>
<tr>
<td>MAX_IN_CONNECT_RATE</td>
<td>Maximum rate at which the Dispatcher has serviced requests from application users</td>
</tr>
</tbody>
</table>
Figure 6.6 shows a query against V$DISPATCHER_RATE comparing the current rate of servicing client requests to the maximum rate for servicing client requests.

**FIGURE 6.6** Current versus maximum Dispatcher servicing rates

```sql
SELECT name, cur_in_connect_rate, max_in_connect_rate,
       cur_in_connect_rate - max_in_connect_rate "VARIANCE"
FROM v$dispatcher_rate;
```

<table>
<thead>
<tr>
<th>NAME</th>
<th>CUR_IN_CONNECT_RATE</th>
<th>MAX_IN_CONNECT_RATE</th>
<th>VARIANCE</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>12</td>
<td>22</td>
<td>10</td>
</tr>
<tr>
<td>0001</td>
<td>6</td>
<td>19</td>
<td>11</td>
</tr>
<tr>
<td>0002</td>
<td>11</td>
<td>28</td>
<td>17</td>
</tr>
</tbody>
</table>

Large variances between current and maximum Dispatcher servicing rates indicate that additional Dispatcher processes may be required to improve Shared Server performance.

The V$SHARED_SERVER_MONITOR dynamic performance view gives an overview of the cumulative activity of the Shared Server environment. This view contains five columns, described in Table 6.7.

**TABLE 6.7** Columns from V$SHARED_SERVER_MONITOR

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAXIMUM_CONNECTIONS</td>
<td>The maximum number of virtual circuits used by the Shared Server options since instance startup</td>
</tr>
<tr>
<td>MAXIMUM_SESSIONS</td>
<td>The maximum number of Shared Server sessions that have been utilized since instance startup</td>
</tr>
<tr>
<td>SERVERS_STARTED</td>
<td>Total number of additional Shared Servers that were automatically started by the Shared Server option since the instance was started—beyond the number of Shared Servers specified by the SHARED_SERVERS init.ora parameter</td>
</tr>
</tbody>
</table>
Figure 6.7 shows a sample query on the V$SHARED_SERVER_MONITOR view.

**TABLE 6.7** Columns from V$SHARED_SERVER_MONITOR (continued)

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SERVERS_TERMINATED</td>
<td>Total number of Shared Servers that were automatically shut down by the Shared Server option since the instance was started</td>
</tr>
<tr>
<td>SERVERS_HIGHWATER</td>
<td>The maximum number of Shared Servers running since the instance was started—both those that were started explicitly in the init.ora and those started implicitly by PMON</td>
</tr>
</tbody>
</table>

The output from the query in Figure 6.7 indicates that two additional Shared Server processes were started by the instance when demands on the three Shared Servers specified by the SHARED_SERVERS init.ora parameter were inadequate to meet the demands of the applications users. One of these two dynamically started Shared Servers was later shut down automatically when demands on the system declined.

Large discrepancies between the values for SERVERS_HIGHWATER and the value for SHARED_SERVERS in the init.ora indicate that the value for SHARED_SERVERS in the init.ora should be increased.

Occasionally you may have a need to trace the path a user’s User Process took from the User Process, to the Dispatcher, to the assigned Shared Server, and back to the User Process again. Figure 6.8 shows how you can use the common columns between the V$CIRCUIT, V$SESSION, V$DISPATCHER, and V$SHARED_SERVER to join the views and construct just such a query.
FIGURE 6.8 Query showing User Process circuit

```
SQL> SELECT s.username, d.name "DISPATCHER", ss.name "SHARED_SERVER"
    2  FROM v$session s, v$circuit c, v$dispatcher d, v$shared_server ss
    3  WHERE s.saddr = c.saddr
    4  AND c.dispatcher = d.paddr
    5  AND c.server = ss.paddr;
```

<table>
<thead>
<tr>
<th>USERNAME</th>
<th>DISP</th>
<th>SHAR</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSTEM</td>
<td>D003</td>
<td>S000</td>
</tr>
<tr>
<td>JOE</td>
<td>D003</td>
<td>S001</td>
</tr>
<tr>
<td>BRENDA</td>
<td>D001</td>
<td>S002</td>
</tr>
<tr>
<td>EMILY</td>
<td>D000</td>
<td>S000</td>
</tr>
<tr>
<td>MATT</td>
<td>D002</td>
<td>S001</td>
</tr>
</tbody>
</table>

Using the output from this query, you can see that Emily’s session is utilizing Dispatcher D000 and Shared Server S000 to service her database requests. This information might be useful for troubleshooting potential performance problems with Emily’s session.

Once the above measures have been used to determine current Dispatcher and Shared Server performance, the adjustments discussed in the next section can be considered for improving the performance of the Shared Pool feature.

Improving Shared Server Performance

Performance problems with the Shared Server architecture are usually related to one of the following areas:

- SGA components are not adequately sized for the Shared Server environment
- Too few Shared Servers running to adequately service user requests
- Too few Dispatchers are running to adequately service user requests

In all of these cases, the tuning solution falls into the “make it bigger” category. Each situation is discussed in the following sections.

Oracle’s Shared Server option should be considered a scale up feature, not a speed up feature. In other words, while the Shared Server feature will allow you to make better use of the available server resources, and thus allow to you support additional application users with those same resources, it will not, at the same time, necessarily improve the overall throughput or response time of the system.
Increase the Size of Related SGA Components

Because Shared Servers operate on a per statement, not per transaction basis, user session and cursor state information (e.g., bind variable values and session settings) normally found in the User Global Area (UGA) inside the user’s Process Global Area (PGA) under the Dedicated Server configuration, are moved to the Shared Pool in the SGA under the Shared Server configuration. As you learned in Chapter 4, the Shared Pool’s primary function is to cache SQL statement and Data Dictionary information. When the caching of Shared Server UGA information is added to the Shared Pool, a decline in Shared Pool hit ratios can result. Thus, at a minimum, the size of the Shared Pool will have to be increased when the Shared Server option is implemented.

Oracle Objective Describe UGA and session memory considerations

The amount by which the Shared Pool should be increased can be determined by querying V$SESSTAT and V$STATNAME dynamic performance views. The following query can be used to determine the maximum amount of memory that has been used to store user session and cursor state information since instance startup:

```
SQL> SELECT SUM(value) 'Total UGA Bytes'
   2   FROM v$sesstat s, v$statname n
   3   WHERE n.name = 'session uga memory max'
   4   AND s.statistic# = n.statistic#;
```

<table>
<thead>
<tr>
<th>Total UGA Bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>365852</td>
</tr>
</tbody>
</table>

In this example, at least 365,852 bytes should be added to the size of the Shared Pool in order to accommodate the UGA information that would be used by the Shared Servers. This additional space can be added to the Shared Pool either dynamically, using the ALTER SYSTEM command, or manually by changing the value of SHARED_POOL_SIZE in the init.ora.

Another solution is to move some of the UGA data out of the Shared Pool and into the Large Pool. See “Configuring the Large Pool” later in this chapter.
Increasing the Number of Shared Servers

Whenever the allocated Shared Servers are unable to service the incoming requests for database activity, the PMON background process will dynamically start additional Shared Server processes, up to the limit specified by the MAX_SHARED_SERVERS value in init.ora. You can also explicitly add additional Shared Server processes either dynamically or manually.

Dynamically Adding Additional Shared Servers

Additional Shared Server processes can be added, up to the maximum specified by the init.ora parameter MAX_SHARED_SERVERS, by using the ALTER SYSTEM command:

```
SQL> ALTER SYSTEM SET shared_servers = 10;
```

This command will dynamically increase the number of Shared Servers to a total of 10.

Manually Adding Additional Shared Servers

Additional Shared Servers can also be added by increasing the value for SHARED_SERVERS in the init.ora. Once the value has been changed, the instance must be shut down and restarted for the changes to take effect.

Increasing the Number of Dispatchers

Unlike Shared Server processes, Dispatcher processes are not dynamically started by the PMON background process when user requests are coming in faster than they can be serviced. If the queries shown in the previous section reveal contention for Dispatchers, you can add additional Dispatchers dynamically or manually.

User Processes remain tied to their original Dispatcher process for the duration of their session. Therefore, any newly added Dispatcher processes go unused by preexisting client connections. Only new connections to the instance will use the new Dispatchers.
Dynamically Adding Additional Dispatchers

Additional Dispatchers can be added, up to the maximum specified by the init.ora parameter MAX_DISPATCHERS, by using the ALTER SYSTEM command:

   SQL> ALTER SYSTEM SET dispatchers = 'tcp, 6';

Both the number of Dispatchers and the protocol associated with the Dispatchers must be specified. In the above example, six TCP/IP Dispatchers were started.

Manually Adding Additional Dispatchers

Additional Dispatchers can also be added by increasing the value for DISPATCHERS in the init.ora. Once the value has been changed, the instance must be shut down and restarted for the changes to take effect.

---

Oracle recommends configuring one Dispatcher process for every 250 concurrent user connections.

Large Pool Concepts

As you saw in Chapter 4, increasing the size of the Shared Pool, setting up a reserved area, and pinning PL/SQL packages are all effective methods of improving the performance of the Shared Pool. However, space reserved for the Shared Pool can also be used for other, non-SQL data. For example, UGA information for the Shared Server option is cached in the Shared Pool. Other optional Oracle features like the Recovery Manager (RMAN) utility and Parallel Query (PQ) can also cause the Shared Pool to cache additional items not related to application SQL.

When these types of optional Oracle features are used, the performance of the Shared Pool can still be negatively impacted even when every effort has been made to pin packages and properly size the Shared Pool and its reserved area.

To solve this problem, Oracle provides the ability to create a special area in the SGA, called the Large Pool, which can be used to process requests for optional features like Recovery Manager and Parallel Query.
Large Pool Concepts

Configuring the Large Pool

To create a Large Pool, set the `init.ora` parameter `LARGE_POOL_SIZE` to the desired size in bytes. By default, this parameter is zero (i.e., no Large Pool exists) unless the `PARALLEL_AUTOMATIC_TUNING` parameter is set to `TRUE`, in which case the size of the Large Pool is set automatically. Once configured, the Large Pool’s memory area can be used to cache data related to:

- I/O slave processes for Database Writer (discussed in Chapter 8, “Tuning Disk I/O”)
- Backup and restore operations for Recovery Manager
- Shared Server session data
- Messaging information for the Parallel Query option

When configuring the Large Pool manually, the minimum non-zero value allowed for `LARGE_POOL_SIZE` is 600K. Once configured, the Oracle Server will automatically use the Large Pool to allocate memory to I/O server processes, RMAN, and Shared Server connections. By doing this, the memory allocated to the Shared Pool will not be consumed by these processes, and will instead be available to application users.

Measuring the Performance of the Large Pool

Once you’ve configured the Large Pool to take advantage of its ability to enhance the performance of the Shared Pool, you should monitor its performance so that you can determine whether it is optimally configured.

You can use the `POOL` column in the dynamic performance view `V$SGASTAT` to track how the space allocated to the Large Pool is being utilized:

```
SQL> SELECT name, bytes 
2  FROM v$sgastat 
3  WHERE pool = 'large pool';
```
The value for free memory shown in this view can be used to help tune the size of the Large Pool. If V$SGASTAT shows high or increasing values for free memory, you have probably overallocated space to the Large Pool. If you have low or decreasing values for free memory, you may need to consider increasing the Large Pool size.

### Improving the Performance of the Large Pool

When warranted by the query shown above, you can increase the size of the Large Pool by increasing the value for LARGE_POOL_SIZE in the init.ora. The size of the Large Pool can be specified in bytes, kilobytes (using the K suffix), or megabytes (using the M suffix). The range of values for LARGE_POOL_SIZE is 600K to 2GB or more depending on operating system. The value of LARGE_POOL_SIZE cannot be changed dynamically using the ALTER SYSTEM command.

---

**Real World Scenario**

**Using the Large Pool to Resolve RMAN Error**

You are running nightly hot backups using the Oracle RMAN utility. You’ve noticed that you are occasionally finding ORA-04031 unable to allocate x bytes of shared memory errors in your Alert Log during the time the backup was running. Additionally, users are complaining of poor application performance during the hot backup period.

When using the Recovery Manager feature in Oracle, RMAN will buffer its I/O activity in the Shared Pool. If database activity is high during the time that the RMAN backup is running, ORA-04031 unable to allocate x bytes of shared memory errors can result. This error indicates that a large piece of contiguous memory was requested in the Shared Pool, but could not be obtained. If this request is coming from RMAN, errors in the backup and poor application performance can result.

When the Large Pool is present, RMAN will use the Large Pool to buffer this I/O instead of the Shared Pool. This will result in not only better RMAN performance but also in improved Shared Pool hit ratios.
Java Pool Concepts

In recent years, application programmers have increasingly turned to Java as the application development platform of choice. The popularity of Java stems from the fact that Java applications are OS independent and can be easily distributed and executed in a web browser–based environment. The Java environment consists of the OS-independent Java application code itself, and the Java Virtual Machine or JVM used to interpret and execute that code on a particular OS. Oracle9i includes several application program interfaces and Java class libraries in order to facilitate the interaction of Java-based applications with Oracle9i databases. Oracle9i also allows you to dedicate a portion of the SGA, called the Java Pool, as the location where session-specific Java code and application variables reside during program execution. Configuring the Oracle9i components correctly is critical to the successful performance of any Java-based Oracle application.

Configuring the Oracle Java Environment

Several init.ora parameters are used to configure the Java environment in Oracle. The primary parameter is JAVA_POOL_SIZE, which is used to set the size of the SGA memory structure where most Java-related session data is cached during execution. Other init.ora parameters related to Java are described in Table 6.8.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SHARED_POOL_SIZE</td>
<td>Shared Pool memory is used by the JVM when Java classes are loaded and compiled and when Java resources in the database are accessed.</td>
</tr>
<tr>
<td>JAVA_POOL_SIZE</td>
<td>All other Java-related session data is cached in the Java Pool memory structure. Default size is 20M; range of values is 1M to 1GB.</td>
</tr>
<tr>
<td>JAVA_SOFT_SESSIONSPACE_LIMIT</td>
<td>When memory is allocated to a Java process, the amount of memory requested is compared to this limit. If the request exceeds this limit, a message is written to a user trace file in the USER_DUMP_DEST. Default value is zero; maximum value is 4GB.</td>
</tr>
</tbody>
</table>
Oracle recommends setting JAVA_POOL_SIZE to 50MB or higher for large Java applications.

**Oracle Objective**

Control the amount of Java session memory used by a session

Using Java, an application programmer can write and load Java applications inside the database. This level of intimacy between the application code and the database is supported because Java applications cannot interact with the OS to the same degree that an OS-specific application can. This separation allows the Java applications to execute safely within the database, with little concern for OS-related security. However, this closeness between Java and the database can also cause serious performance problems if an errant Java application consumes too many database resources. The JAVA_SOFT_SESSIONSPACE_LIMIT and JAVA_MAX_SESSIONSPACE_SIZE described in Table 6.8 are intended to let you limit the resources that a Java session is allowed to utilize within the database.

**Measuring the Performance of the Java Pool**

Once you’ve configured the Java Pool to take advantage of its ability to enhance the performance of the Java-based database applications, you should monitor its performance so that you can determine whether it is optimally configured. You can use the V$SGASTAT dynamic performance view and STATSPACK output to monitor the performance of the Java Pool.
Measuring Java Pool Performance Using V$SGASTAT

You can use the POOL column in the dynamic performance view V$SGASTAT to track how the memory allocated to the Java Pool is being utilized:

```
SQL> SELECT name, bytes
2  FROM v$sgastat
3  WHERE pool = 'java pool';
```

<table>
<thead>
<tr>
<th>NAME</th>
<th>BYTES</th>
</tr>
</thead>
<tbody>
<tr>
<td>free memory</td>
<td>58720256</td>
</tr>
<tr>
<td>memory in use</td>
<td>277821</td>
</tr>
</tbody>
</table>

The value for free memory shown in this view can be used to help tune the size of the Java Pool. If V$SGASTAT shows high or increasing values for free memory when compared to memory in use, you have probably overallocated space to the Java Pool. If you have low or decreasing values for free memory, you may need to consider increasing the Java Pool size.

Measuring Java Pool Performance Using STATSPACK

The output from the STATSPACK utility also includes information of the performance of the Java Pool. Figure 6.9 shows sample output for this section of the STATSPACK report.

![STATSPACK output showing Java Pool statistics](image)

The output shown in Figure 6.9 shows that the free memory in the Java Pool started at 58MB at the beginning of the STATSPACK monitoring.
period, but declined to approximately 19MB by the end of the snapshot period. Ending Java Pool values that are near or steadily approaching zero indicate that the Java Pool may be too small.

**Improving the Performance of the Java Pool**

When warranted by the query shown above, you can increase the size of the Java Pool by increasing the value for JAVA_POOL_SIZE in init.ora. The size of the Large Pool can be specified in bytes, kilobytes (using the K suffix), or megabytes (using the M suffix). The range of values for JAVA_POOL_SIZE is 1M to 1GB. The size of the Java Pool cannot be changed dynamically.

If the Large Pool is configured, Java operations will make use of the Large Pool to store UGA information.

**Summary**

Oracle9i includes several features that, if utilized, will impact the overall performance of the SGA structures. The Shared Server option is designed to allow you to add additional application users to an existing system without having to add additional CPU or memory resources to support those users. The Shared Server configuration accomplishes this by using Dispatchers and queues to juggle incoming user requests between several available Shared Server processes instead of dedicating a Server Process to each user. Shared Servers and Dispatchers can both be monitored using dynamic performance views in order to determine whether there are enough of them available to service the user requests adequately. Additional Shared Server and Dispatcher processes can be started either manually or dynamically.

The Large Pool can be used to buffer UGA information for the Shared Server configuration as well as I/O related data for the RMAN, Parallel Query, and Database Writer I/O slaves options. The effectiveness of the Large Pool can also be analyzed by examining dynamic performance views, and its size adjusted accordingly.
The Java Pool is an SGA component used to cache Java state information from users accessing the database via Java-based applications. Like the Large Pool, the performance of the Java Pool can also be determined by examining dynamic performance views, and its size adjusted accordingly.

Exam Essentials

Understand the Shared Server Architecture. Be able to describe how the optional Shared Server architecture differs from the default Dedicated Server configuration. Know the name and purpose of each Shared Server component and how to measure its performance.

Know the Role of the Large Pool. Know which init.ora parameters are used to configure the Large Pool and under what circumstances the Large Pool should be used to enhance database performance. Be able to determine the effectiveness of the Large Pool's performance.

Describe the Functionality of the Java Pool. Know which init.ora parameters are used to configure the Java Pool and constrain the resources consumed by any run-away Java processes. Understand under what circumstances the Java Pool is needed and how to determine its effectiveness in terms of performance.

Key Terms

Before you take the exam, be certain you are familiar with the following terms:

- busy ratio
- Dedicated Server
- Dispatcher
- Java Pool
- Java Virtual Machine
- Large Pool
- Request Queue
- Response Queue
- Shared Server
Review Questions

1. Which two of the following options are possible ways in which a client can connect to an Oracle9i database?
   A. Shared Server
   B. Dynamic Server
   C. Dedicated Server
   D. Multithreaded Server

2. The following are components of the Oracle Shared Server Architecture except:
   A. Dispatchers
   B. Shared Servers
   C. Request Queue
   D. Reply Queue

3. What are the background processes that run on the Oracle server machine and accept incoming requests for database processing in a Shared Server environment called?
   A. User Processes
   B. Dispatchers
   C. Shared Servers
   D. Dedicated Servers

4. Several conditions must be met before a Shared Server configuration can be implemented. These include all of the answers below except which one?
   A. Many application users
   B. Short application transactions
   C. Noncontinuous transactions
   D. Continuous transactions
5. Implementing a Shared Server architecture would address which of the following problems?
   A. I/O bottlenecks
   B. Maximum number of processes on the machine being reached
   C. CPU bottlenecks
   D. Application SQL problems

6. What is the queue into which the Dispatcher places messages received from User Processes called?
   A. Request Queue
   B. Message Queue
   C. Dispatcher Queue
   D. Response Queue

7. What is the parameter that is used to specify the maximum number of Shared Server Processes called?
   A. SHARED_SERVERS_MAX
   B. MAXIMUM_SERVERS
   C. MAX_SHARED_SERVERS
   D. SHARED_SERVERS

8. Which of the following commands could not be performed while connected to a Shared Server?
   A. ALTER DATABASE
   B. RECOVER DATABASE
   C. ALTER SYSTEM
   D. ALTER TABLESPACE ABC BEGIN BACKUP

9. “Contains detailed statistics about the Shared Server Dispatcher processes, including how actively the Dispatchers have been used for processing” describes what dynamic performance view?
Tuning Other SGA Areas

A. V$SHARED_SERVER
B. V$DISPATCHER
C. V$MTS
D. V$DISPATCHER_PROCESS

10. The DBA wants to increase the number of Shared Server processes to 10. What would be the quickest way to perform this?
   A. ALTER SYSTEM SET shared_servers = 10;
   B. Stop the database. Change the SHARED_SERVERS parameter in the init.ora file and restart.
   C. ALTER SESSION SET shared_servers = 10;
   D. ALTER DATABASE SET shared_servers = 10;

11. Which of the following statements is true about Shared Servers?
   A. They place completed information into a Request Queue.
   B. They receive requests from Client processes.
   C. They can respond to requests from multiple dispatchers.
   D. They operate on a Per Transaction basis.

12. The Large Pool can be used for all of the following except:
   A. I/O slave processes for Database Writer
   B. Buffering Redo Log entries
   C. Shared Server session data
   D. Messaging information for the Parallel Query option

13. The Large Pool is configured using what Oracle parameter?
   A. LARGE_POOL_SIZE
   B. LARGE_POOL
   C. LARGE_POOL_CACHE
   D. LARGE_POOL_BUFFER
14. What effect does the PARALLEL_AUTOMATIC_TUNING parameter have on the Large Pool?
   A. None
   B. Causes it not to be used
   C. Causes it to be configured dynamically
   D. Enables the ability to use the large pool

15. What Dynamic performance view can be used to monitor the size of the Large Pool?
   A. V$LARGE_POOL
   B. V$SGASTAT
   C. V$POOLSTAT
   D. V$LARGE_POOL_SIZE

16. What is the minimum size of the Large Pool?
   A. 128K
   B. 4M
   C. 16M
   D. 600K

17. What is the name of the place where session-specific Java code and application data reside during program execution?
   A. Large Pool
   B. Java Pool
   C. Shared Pool
   D. Database Buffer Cache

18. What parameter is used to configure the JAVA_POOL?
   A. JAVA_POOL
   B. JAVA_SHARED_POOL
   C. JAVA_POOL_SIZE
   D. SHARED_POOL_SIZE
19. You can monitor the amount of space allocated by a user session in the Java Pool by setting what parameter?
   A. JAVA_SPACE_LIMIT
   B. JAVA_SOFT_SESSION_LIMIT
   C. JAVA_SOFT_LIMIT
   D. JAVA_SOFT_SESSIONSPACE_LIMIT

20. What is the default value of the JAVA_POOL_SIZE?
   A. 128K
   B. 600K
   C. 8M
   D. 20M
Answers to Review Questions

1. A, C. Clients can connect to an Oracle9i database via a Dedicated Server connection or via a Shared Server connection. Be careful on this one because Multithreaded Server is the old name for the Shared Server.

2. D. Dispatchers, Shared Servers, and the Request Queue are all components of the Oracle Shared Server Configuration. There is no such component as the Reply Queue.

3. B. Dispatcher processes are responsible for interacting with the client processes and place client requests in the Request Queue where the Shared Servers picks up and processes the request.

4. D. The Shared server configuration is a scalability solution. The types of applications that this is suited for are characterized by all of the above except continuous transactions.

5. B. In a Shared Server environment, clients share dispatchers. Therefore many clients can be serviced by relatively few dispatchers. This solution would alleviate situations where a machine is hitting its maximum number of processes.

6. A. The Dispatcher places client requests into the Request Queue where the Shared Server Processes will pick them up and process them. The completed requests are then placed in the Response Queue.

7. C. The MAX_SHARED_SERVERS parameter specifies the maximum number of Shared Server Processes that can be started. This number can be adjusted if more Shared Servers are needed.

8. B. A database recovery of this type would have to be done using a dedicated connection.


10. A. The number of shared servers can be dynamically set by issuing the ALTER SYSTEM command. You could also stop the database and reset the SHARED_SERVERS parameter, but this would not be the quickest way to change the parameter and would require system downtime.
11. C. Shared servers can process requests from any of the dispatchers. In this way, Oracle maximizes the efficiency of the Shared Server processes.

12. B. Redo log entries are buffered in the Redo Log Buffer, not the Large Pool.

13. A. The LARGE_POOL_SIZE initialization parameter is used to set the size of the Large Pool.

14. C. If this parameter is set, then LARGE_POOL_SIZE will be configured automatically.

15. B. You would query the V$SGASTAT and look for pool = 'large pool'.

16. D. The default size of the Large Pool is 600K. You can override this by setting the LARGE_POOL_SIZE parameter in the init.ora file.

17. B. The Java Pool, introduced in Oracle 8i, is the location in the SGA where Oracle will store recently executed Java Stored Procedure code.

18. C. The Java pool is configured by setting the JAVA_POOL_SIZE parameter in the init.ora file.

19. D. Setting JAVA_SOFT_SESSIONSPACE_LIMIT to a specific value will cause Oracle to write warning messages to user trace files whenever a user's Java memory usage exceeds the specified value. You may want to consider using this parameter if your application makes extensive use of Java Stored Procedures.

20. D. The default size of the Java Shared Pool JAVA_POOL_SIZE parameter is 20M.
Chapter 7

Tuning Redo Mechanisms

ORACLE9i PERFORMANCE TUNING EXAM OBJECTIVES COVERED IN THIS CHAPTER:

- Monitor and size the Redo Log Buffer
- Monitor and tune Redo Logs
- Describe how checkpoints work
- Monitor and tune checkpoints
- Describe performance and safety tradeoffs

Exam objectives are subject to change at any time without prior notice and at Oracle’s sole discretion. Please visit Oracle’s Certification website (http://www.oracle.com/education/certification) for the current exam objectives listing.
In this chapter, you will see how the role of the last required component of the SGA, the Redo Log Buffer, complements the functionality of the two SGA structures you have already explored in Chapters 4 (“The Shared Pool”) and 5 (“Tuning the Database Buffer Cache”). This chapter will discuss the role of the Redo Log Buffer, its mechanisms, how to monitor its performance, and how to tune it when needed. This chapter will also discuss tuning the physical components of Oracle’s redo mechanisms, including Redo Log I/O and the Redo Log archiving process.

Understanding Oracle’s Redo Mechanisms

Oracle’s redo mechanisms record the changes made to application data by application users so that those committed transactions can be “redone” in the event of a server hardware failure or user error. Oracle’s redo mechanisms consist of five components:

- Redo Log Buffer
- Log Writer (LGWR)
- Database checkpoints
- Online Redo Log files
- Archived Redo Log files

**Redo Log Buffer**  The Redo Log Buffer is the portion of the SGA that records the information needed to redo an application user’s transactions in the event of media failure or user error. These transactions may be DML statements like `INSERT`, `UPDATE`, or `DELETE`, or DDL statements like `CREATE`, `ALTER`, or `DROP`. Redo information is copied into the Redo Log Buffer by each user’s Server Process.
Log Writer (LGWR)  The Log Writer background process is responsible for writing the contents on the Redo Log Buffer to the Online Redo Logs. Each instance has only one Log Writer.

Database Checkpoints  A database checkpoint event represents the moment in time when the database is in a consistent state. Checkpoints are important because, following instance failure, only those transactions that occurred after the last checkpoint have to be recovered during instance recovery.

Online Redo Logs  When the Log Writer (LGWR) background process writes the contents of the Redo Log Buffer to disk, these entries are written to physical files called Redo Logs. Every database must have a minimum of at least two Redo Log files. When LGWR fills the current Redo Log with redo information, LGWR switches to the next Redo Log group before writing the next batch of redo information.

Archived Redo Logs  After LGWR has written to all the available Online Redo Logs, LGWR will start over by writing to the first Redo Log group again. If the data in this Redo Log is not preserved somehow, the redo information contained in the log will be lost—limiting your ability to perform a complete database recovery. However, a database operating in Archive Log Mode will not reuse an Online Redo Log until the Archiver (ARC0) background process has copied the contents of the log to the archive destination.

Figure 7.1 illustrates the concept of the Redo Log Buffer mechanism.

| Oracle Objective | Describe performance and safety tradeoffs |

Some of Oracle’s optional redo mechanisms, like redo log archiving and standby databases, can hinder performance because of the additional I/O activity they cause. However, any potential decline in performance resulting from the use of these features must be carefully weighed against the greatly improved recoverability these features will provide in the event database disaster recovery is required. Operating a production database in non-Archive Log Mode just to avoid additional I/O is not advised.

Log Writer generally does a pretty good job of keeping up with managing the contents of the Redo Log Buffer. However, the following sections discuss techniques for monitoring and tuning these redo components when redo bottlenecks occur.
Unlike the Shared Pool and Database Buffer Cache, the Redo Log Buffer is not managed by an LRU mechanism. Instead of the conveyor belt concept of the LRU List, the Redo Log Buffer’s management mechanism can be thought of as funnel, where transaction redo information enters at the top of the funnel and is then occasionally emptied out the bottom of the funnel by the LGWR background process. To keep up with the volume of incoming Redo Log Buffer entries, LGWR is prompted to write the contents of the Redo Log Buffer to disk whenever any of the following events occur:

- Whenever an application user issues a COMMIT command
- Every three seconds
Whenever the Redo Log Buffer is one-third full
- Whenever the volume of redo entry information in the Redo Log Buffer reaches 1MB
- Whenever a database checkpoint occurs

The size of the Redo Log Buffer is determined by the init.ora parameter LOG_BUFFER.

With each I/O, LGWR writes all the redo information that has been added to the Redo Log Buffer since the last LGWR write operation.

**Measuring the Performance of the Redo Log Buffer**

One way the performance of the Redo Log Buffer can be measured is in terms of the number and length of waits that user Server Processes experience when trying to place entries in the Redo Log Buffer. These waits can be viewed in the V$SYSSTAT and V$SESSION_WAIT dynamic performance views. These same statistics can also be found in STATSPACK output and the REPORT.TXT file generated by UTLBSTAT.SQL and UTLESTAT.SQL.

### Oracle Objective

- **Monitor and size the Redo Log Buffer**

**Using V$SYSSTAT to Measure Redo Log Buffer Performance**

The database statistics recorded in V$SYSSTAT that are relevant to tuning the Redo Log Buffer are redo buffer allocation retries, redo entries, and redo log space requests.

The statistic redo entries reports the number of entries that have been placed into the Redo Log Buffer by the user Server Processes since instance startup. The statistic redo buffer allocation retries refers to the number of times user Server Processes had to wait and then retry placing their entries in the Redo Log Buffer because LGWR had not yet written the current entries to the Online Redo Log. Using these two statistics, it is possible
to calculate a Redo Log Buffer Retry Ratio using the following query on 
V$SYSSTAT:

```
SQL> SELECT retries.value/entries.value "Redo Log Buffer Retry Ratio"
2   FROM   v$sysstat retries, v$sysstat entries
3   WHERE  retries.name = 'redo buffer allocation retries'
4   AND    entries.name = 'redo entries';
```

Redo Log Buffer Retry Ratio
----------------------------

.000345

Ideally you would like the user Server Processes to never have to wait for 
access to the Redo Log Buffer. High or increasing values for this ratio indicates 
that the Redo Log Buffer may need tuning.

Oracle recommends that this Redo Log Buffer Retry Ratio should be less 
than 1 percent.

The statistic redo log space requests in V$SYSSTAT measures how 
often LGWR is waiting for a Redo Log switch to occur when moving from 
the current Online Redo Log to the next:

```
SQL> SELECT name, value
2   FROM v$sysstat
3   WHERE name = 'redo log space requests';
```

<table>
<thead>
<tr>
<th>name</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>redo log space requests</td>
<td>34</td>
</tr>
</tbody>
</table>

During a Redo Log switch, LGWR cannot empty the contents of the Redo 
Log Buffer. This may cause user Server Processes to experience a wait when 
trying to access the Redo Log Buffer. This wait generates a retry request as 
explained in the previous section. Therefore, high or increasing values for 
redo log space requests indicate that your Redo Logs may be too small 
and thus contributing to the Redo Log Buffer Retry Ratio. This can also indicate 
a problem with I/O contention. Tuning physical I/O is discussed in the 
next chapter.
See Appendix C, “Statistic Descriptions,” in Oracle9i Database Reference, Release 1 (Part Number A90190-01) for definitions of all the statistics found in V$SYSSTAT.

### Real World Scenario

**Sizing the Redo Log Buffer**

Imagine you’ve inherited an existing Oracle database that is used to support your company’s order-taking system. After examining the Shared Pool and Database Buffer Cache hit ratios, you realize that they could probably be improved by making these structures larger, but there is not sufficient memory in the server to do so. The Redo Log Buffer’s Retry Ratio is low—0.0000012.

Before you give up and ask the boss to order more memory for the server, examine the size of the Redo Log Buffer. Chances are, with a Retry Ratio this low, the Redo Log Buffer may be oversized and wasting memory.

Suppose you check the size of the Redo Log Buffer and discover that it is 10MB. Any value above 1MB will be largely unused because LGWR is signaled to write when the Redo Log Buffer is one-third full or filled with 1MB of redo entries. By reducing the size of the Redo Log Buffer to 1MB or less, you can allocate the approximately 9MB remaining space to the Shared Pool and/or Database Buffer Cache. This will likely improve the overall SGA hit ratios without consuming any additional memory on the server.

### Using V$SESSION_WAIT to Measure Redo Log Buffer Performance

Unlike V$SYSSTAT, which shows statistics for the entire instance, V$SESSION_WAIT shows how long, and for which events, individual user sessions have waited. The log buffer space statistic indicates how long the user session had to wait to place an entry in the Redo Log Buffer because LGWR had not yet finished writing the contents of the Redo Log Buffer to the Online Redo
Log. You can use the following query on both V$SESSION_WAIT and V$SESSION to monitor this activity on a per-user basis:

```
SQL> SELECT username, wait_time, seconds_in_wait
2   FROM v$session_wait, v$session
3   WHERE v$session_wait.sid = v$session.sid
4   AND event LIKE '%log buffer space%';
```

<table>
<thead>
<tr>
<th>USERNAME</th>
<th>SECONDS_IN_WAIT</th>
<th>STATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>JOE</td>
<td>20</td>
<td>WAITING</td>
</tr>
<tr>
<td>MATT</td>
<td>213</td>
<td>WAITED SHORT TIME</td>
</tr>
</tbody>
</table>

Table 7.1 shows the possible values for `STATE` in V$SESSION_WAIT and the descriptions for each of these states.

**Table 7.1** Column Values and Descriptions for `STATE` in V$SESSION_WAIT

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WAITING</td>
<td>The session is waiting at that moment.</td>
</tr>
<tr>
<td>WAITED UNKNOWN TIME</td>
<td>Duration of the last wait is undetermined.*</td>
</tr>
<tr>
<td>WAITED SHORT TIME</td>
<td>The last wait lasted less than 1/100th of a second.</td>
</tr>
<tr>
<td>WAITED KNOWN TIME</td>
<td>The value in the WAIT_TIME column is the actual wait time.</td>
</tr>
</tbody>
</table>

*On operating systems that do not support a fast timing mechanism, this value may not always be displayed until the init.ora parameter TIMED_STATISTICS is set to TRUE.*

High or increasing values for waits related to log buffer space may indicate that the Redo Log Buffer needs to be tuned.

See Appendix A, “Oracle Wait Events,” in *Oracle9i Database Reference, Release 1* (Part Number A90190-01) for definitions of all the statistics found in V$SESSION_WAIT and V$SYSSTAT.
Using STATSPACK Output to Measure Redo Log Buffer Performance

The output from the STATSPACK feature also includes some information related to the performance of the Redo Log Buffer. This information appears in two different sections of the STATSPACK output file.

The first section, headed *Instance Efficiency Percentages*, indicates the performance of the Redo Log Buffer in terms of a ratio called the *Redo NoWait*%. This is the converse of the Redo Log Buffer Retry Ratio shown in the previous query against V$SYSSTAT. Instead of showing what percentage of the time user Server Processes experienced a wait when accessing the Redo Log Buffer, the STATSPACK output shows the percentage of time that user Server Processes did *not* experience a wait when attempting to place an entry in the Redo Log. Figure 7.2 shows this section of a sample STATSPACK output file.

**FIGURE 7.2** STATSPACK output showing Redo Log Buffer performance

```
Instance Efficiency Percentages (Target 100%)
---------------------------------------------
Buffer NoWait %: 100.00  Redo NoWait %: 99.12
Library Hit %: 96.11  In-memory Sort %: 97.96
Execution to Parse %: 42.56  Soft Parse %: 76.01
Parse CPU to Parse Elapsed %: % Non-Parse CPU:
```

This output shown in Figure 7.2 indicates that 99.12 percent of the time, user Server Processes did not have to wait to place their entries into the Redo Log Buffer.

**TIP**

Oracle recommends that this Redo Log Buffer No Wait Ratio should be 99 percent or better.

The second section of the STATSPACK output that shows Redo Log Buffer performance information is headed *Instance Activity Stats*. This section contains the same statistical information found in the V$SYSSTAT dynamic performance view. Figure 7.3 shows this section of a sample STATSPACK output file.

The output shown in Figure 7.3 indicates that every attempt by a user Server Process to place an entry in the Redo Log Buffer (redo entries) succeeded without having to wait (redo buffer allocation retries) for access to the Redo Log Buffer.
Using REPORT.TXT Output to Measure Redo Log Buffer Performance

The REPORT.TXT file generated by running UTLBSTAT.SQL and UTLESTAT.SQL also contains a section that shows the same statistics related to Redo Log Buffer performance that are found in V$SYSSTAT and STATSPACK output. The relevant sections of a sample REPORT.TXT are shown in Figure 7.4.

```
SQL> select n1.name "Statistic", n2.change "Total", round(n2.change/trans_change,2) "Per Transaction", round(n1.change/(t1."start_users" + end_users)/2,2) "Per Logon", round(n2.change/to_number(to_char(t1."start_time", 'HH24:MISS')) - to_number(to_char(t1."end_time", 'HH24:MISS')))*86400/80/24 "Per Second" from stats$r$stat rd, stats$r$stat trans, stats$r$dates where trans$name = 'user commits' and rd.frame = 0 order by rd.name;
```
The output shown in Figure 7.4 shows that out of the 271,852 redo entries (redo entries) that were placed in the Redo Log Buffer between the times UTLBSTAT.SQL and UTLESTAT.SQL were executed, 307 entries (redo buffer allocation retr) experienced a wait and had to re-attempt placing their entry in the Redo Log Buffer.

The names of some statistics are truncated in the output for REPORT.TXT. For example, the statistic redo buffer allocation retry is truncated to redo buffer allocation retr.

Improving Redo Log Buffer Performance

Now that you know how the Redo Log Buffer mechanism works and how to monitor its performance, you can examine methods for improving its performance. The objective of Redo Log Buffer tuning is to make sure that user Server Processes have access to, and can place entries in, the Redo Log Buffer without experiencing a wait. These techniques fall into two categories:

- Make it bigger
- Reduce redo generation

Oracle Objective

Monitor and size the Redo Log Buffer

Make It Bigger

The easiest way to improve the performance of the Redo Log Buffer is to increase its size. The larger the Redo Log Buffer is, the less likely user Server Processes are to experience a wait when trying to place redo entries into the buffer.

The size of the Redo Log Buffer is specified in bytes using the init.ora parameter LOG_BUFFER. The default value for LOG_BUFFER is 512KB, or 128KB \times \text{the number of CPUs in the server}, whichever is greater. The range of values is 64K to an OS specific maximum.

If you make the size of the Redo Log Buffer too small by setting LOG_BUFFER to a very low value, the Oracle Server will override your settings and increase the Redo Log Buffer to its default size.
As with the Shared Pool and the Database Buffer Cache, it is an acceptable tuning practice to continue increasing the size of the Redo Log Buffer until the Redo Log Buffer Retry Ratio shows no further improvement or the value for the statistic log buffer space in V$SESSION_WAIT ceases to diminish.

Reduced contention for the Redo Log Buffer latches is a secondary indicator of improved Redo Log Buffer performance. Latch tuning is discussed in Chapter 9, “Tuning Contention.”

Reduce Redo Generation

Another way to improve the performance of the Redo Log Buffer is to reduce the demands placed upon it by reducing the amount of redo information generated by certain DML statements. This technique is implemented using one of the following methods:

- UNRECOVERABLE keyword
- NOLOGGING keyword

UNRECOVERABLE Keyword

The UNRECOVERABLE keyword is used when creating a table using the CREATE TABLE AS SELECT SQL command:

```
SQL> CREATE TABLE employee_history
    2  AS
    3  SELECT *
    4  FROM employee
    5  UNRECOVERABLE;
```

Tables created in this manner do not generate any redo information for the inserts generated by the CREATE statement’s sub-query. However, any subsequent DML on the table will generate entries in the Redo Log Buffer. By default, all tables created using the CREATE TABLE AS SELECT syntax are recoverable.

The UNRECOVERABLE/RECOVERABLE keyword cannot be used when creating partitioned tables, index-organized tables, or tables containing Large Objects (LOBS). For this reason, it has been largely replaced by the NOLOGGING option since Oracle version 8.0.x.
**NOLOGGING** Keyword

Like **UNRECOVERABLE**, the **NOLOGGING** option also allows you to specify which tables should skip generating redo entries when certain types of DML are performed on them. However, unlike **UNRECOVERABLE**, **NOLOGGING** is an attribute of the table that stays in effect until you disable it. You can specify the **NOLOGGING** attribute at either table creation, or after creation using the **ALTER TABLE** statement.

**NOLOGGING during Table Creation**  The following example shows how the **NOLOGGING** keyword can be specified during table creation:

```sql
SQL> CREATE TABLE employee_history
2   (employee_id number,
3   last_name varchar2(30),
4   first_name varchar2(30),
5   hired_date date,
6   start_date date,
7   end_date date)
8   TABLESPACE appl_tab
9   STORAGE (INITIAL 500k NEXT 500k PCTINCREASE 0)
10  NOLOGGING;
```

**NOLOGGING after Table Creation**  The **NOLOGGING** option can also be specified in the **ALTER TABLE** command:

```sql
SQL> ALTER TABLE employee_history NOLOGGING;
```

You can also alter a table back into **LOGGING** mode with the **ALTER TABLE** command:

```sql
SQL> ALTER TABLE employee_history LOGGING;
```

All tables are **LOGGING** tables unless the **NOLOGGING** option is specified at either table creation or by using the **ALTER TABLE** command.

You can determine which tables have the **NOLOGGING** attribute by querying **DBA_TABLES**:

```sql
SQL> SELECT owner, table_name
2   FROM dba_tables
3   WHERE logging = 'NO';
```
Once the NOLOGGING attribute is set on a table, redo entry generation will be suppressed for all subsequent DML on the table only when that DML is of the following types:

- Direct Path loads using SQL*Loader
- Direct load inserts using the */ APPEND */ hint

Any subsequent DML on a NOLOGGING table that does not fall into one of these two categories will still generate regular redo entries in the Redo Log Buffer.

In addition to the CREATE TABLE and ALTER TABLE commands shown above, the following commands can also be used with the NOLOGGING option:

- CREATE TABLE AS SELECT
- CREATE INDEX
- ALTER INDEX REBUILD
- CREATE TABLESPACE

Specifying the NOLOGGING attribute on a table means that any data that was loaded using the direct path methods will be lost if media recovery is required before the new data that was added has been included in a scheduled cold or hot database backup.

**Tuning Checkpoints**

The Log Writer background process writes the contents of the Redo Log Buffer to the Online Redo Log whenever a database checkpoint event occurs in the database.
A database checkpoint event represents the moment in time when the database is in a consistent state. Checkpoints are important because, following instance failure, only those transactions that occurred after the last checkpoint have to be recovered during instance recovery. There are two types of checkpoints: *incremental* and *full*. When an incremental checkpoint occurs, the following actions take place:

- The `CKPT` background process updates the control file headers immediately.
- The `CKPT` background process updates the control file headers (again) and datafile headers at the new Redo Log switch.

When a full checkpoint event occurs, the following actions occur in the database:

- All buffers in the Database Buffer Cache that contain committed transactions are written to disk by the `DBWR` background process.
- All the contents of the Redo Log Buffer are written to the Online Redo Log by the LGWR background process.
- Database control files and datafile headers are updated by the `CKPT` background process to indicate that the checkpoint event has occurred.

Database checkpoints occur in the database whenever:

- The instance is shutdown using any method except ABORT.
- When the Online Redo Log switches from the current log to the next Redo Log in the sequence.
- When the DBA issues the `ALTER SYSTEM CHECKPOINT` command.
- When a tablespace is placed into hot backup mode using the `ALTER TABLESPACE ... BEGIN BACKUP` command or when a tablespace is taken offline. This checkpoint is a special form of the complete checkpoint referred to as a full tablespace checkpoint. During this type of checkpoint, only the dirty buffers belonging to the tablespace in hot backup mode are written to disk.
When the value specified by the `init.ora` parameter `FAST_START_MTTR_TARGET` is exceeded.

It is important to monitor checkpoint activity because having *too many* checkpoints causes unnecessary I/O and having *too few* checkpoints exposes the database to protracted instance recovery times. The following section describes techniques for measuring the performance of database checkpoints.

### Measuring the Performance of Checkpoints

Achieving the proper balance between the I/O overhead associated with the checkpoint event and its instance recovery benefits requires that checkpoint activity be monitored using the contents of the `V$SYSTEM_EVENT` and `V$SYSSTAT` dynamic performance views, STATSPACK, `REPORT.TXT`, the Alert log, and the OEM Performance Manager.

### Oracle Objective

Monitor and tune checkpoints

### Using `V$SYSTEM_EVENT` to Measure Checkpoint Performance

Occasionally, checkpoint events are started, but do not complete successfully because a second checkpoint request occurs very soon after the first. You can detect checkpoints that are started but not completed by querying the statistic log file switch (checkpoint incomplete) in `V$SYSTEM_EVENT`. This dynamic performance view reports the number of waits that have occurred since instance startup for a variety of events. Table 7.2 describes the contents of the `V$SYSTEM_EVENT` columns used in this query.

### Table 7.2

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EVENT</td>
<td>The name of the event associated with the wait</td>
</tr>
<tr>
<td>TOTAL_WAITS</td>
<td>The total number of times a wait occurred for this event</td>
</tr>
<tr>
<td>AVERAGE_WAIT</td>
<td>The average time, in 100ths of a second, spent waiting for this event</td>
</tr>
</tbody>
</table>
Figure 7.5 shows a sample query on V$SYSTEM_EVENT showing the waits related to checkpoint activities that have occurred in the database.

**FIGURE 7.5** Checkpoint activity shown in V$SYSTEM_EVENT

```sql
SQL> SELECT event, total_waits, average_wait
2    FROM V$SYSTEM_EVENT
3    WHERE event LIKE '%checkpoint%'
4    OR event LIKE '%log file switch';
```

<table>
<thead>
<tr>
<th>EVENT</th>
<th>TOTAL_WAITS</th>
<th>AVERAGE_WAIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>checkpoint completed</td>
<td>1</td>
<td>20</td>
</tr>
<tr>
<td>log file switch (checkpoint incomplete)</td>
<td>54</td>
<td>38</td>
</tr>
</tbody>
</table>

Two events found in the V$SYSTEM_EVENT view are indicators of checkpoint performance: checkpoint completed and log file switch (checkpoint incomplete).

**Checkpoint Completed** This event shows how often waits occurred for the checkpoint to complete its activities. High or steadily increasing values for this event indicate that checkpoints need to be tuned. The output in Figure 7.5 shows that one Checkpoint Complete event had to wait 10 hundredths of a second to complete.

**Log File Switch (Checkpoint Incomplete)** This event shows how often the Online Redo Log switched from one log to the next, before the checkpoint from the previous log switch had time to complete. When this occurs, the in-progress checkpoint is abandoned and a new checkpoint is begun. Because incomplete checkpoints cause excess I/O that do not provide any recovery benefits, frequent occurrences of this event indicate that checkpoint activity should be tuned. The output from Figure 7.5 shows that log switches that lead to incomplete checkpoints occurred 54 times, experiencing an average wait of 38 hundredths of a second.

Another indicator of checkpoint performance can be found in the V$SYSSTAT dynamic performance view. Two statistics, background checkpoints started and background checkpoints completed, can be used to determine whether all the checkpoints that were started, actually completed. Discrepancies between these two statistics indicate that checkpoints are starting, but not completing. As with the log file switch (checkpoint incomplete) event, this situation occurs anytime checkpoints are occurring too closely together. Figure 7.6 shows an example of this problem using the V$SYSSTAT dynamic performance view.
FIGURE 7.6 Checkpoint activity shown in V$SYSSTAT

The output in Figure 7.6 shows that 36 checkpoints were started and 36 checkpoints completed successfully. Any time the difference between these two values is greater than 1, checkpoint tuning should be considered.

The number of background checkpoints started and the number of background checkpoints completed will occasionally differ by one checkpoint if the query is executed when a checkpoint is in progress. This does not necessarily indicate a checkpoint performance problem.

Using STATSPACK and REPORT.TXT to Measure Checkpoint Performance

The checkpoint activity shown in the query in Figure 7.6 is also contained in the output from the STATSPACK option and REPORT.TXT files. Figure 7.7 shows a sample of this section of a STATSPACK report.

FIGURE 7.7 Checkpoint activity shown in STATSPACK output

Figure 7.8 shows sample output from a REPORT.TXT file that includes checkpoint information.
The output in both these figures shows that while 22 checkpoints were started, only 17 of those checkpoints ran to completion. This indicates that 4 or 5 of these checkpoints did not complete because they were interrupted by a subsequent checkpoint event. This is an indication that this instance should have its checkpoint activity tuned.

Using the Alert Log to Measure Checkpoint Performance

The Alert log also contains important information regarding checkpoint activity. When found in the Alert log, the following messages indicate that the Redo Logs are switching too fast and not allowing enough time for a checkpoint that has been started to complete normally:

- Thread 1 advanced to log sequence 293
- Current log# 1 seq# 293 mem# 0: /u03/oradata/PROD/log1a.dbf
- Current log# 1 seq# 293 mem# 1: /u04/oradata/PROD/log1b.dbf
- Thread 1 cannot allocate new log, sequence 294
- Checkpoint not complete
Even more verbose checkpoint information will also be recorded in the Alert log when the LOG_CHECKPOINTS_TO_ALERT is set to TRUE in the init.ora.

**Using Performance Manager to Measure Checkpoint Performance**

The Performance Manager component of OEM can also be used to monitor the performance of checkpoint activity. The Average Dirty Queue Length option of the Performance Manager I/O feature indicates how often checkpoint activity is occurring. Figure 7.9 shows the GUI output from a sample Average Dirty Queue Length screen.

**FIGURE 7.9** OEM Performance Manager Average Dirty Queue Length

The output in Figure 7.9 shows that the Average Dirty Queue Length was close to zero throughout the monitoring period. This indicates that checkpoint activity occurred fairly frequently during the monitoring period. High or steadily increasing values for Average Dirty Queue Length indicate a potential checkpoint-related performance bottleneck.

**Improving Checkpoint Performance**

As described in the previous section, the checkpoint event is an I/O intensive moment in the database. Once a checkpoint is started, you would like it to complete successfully. If it does not, the checkpoint event ends up causing unnecessary I/O by prompting DBW0 and LGWR to empty the Database
Buffer Cache and Redo Log Buffer, but provides no recovery benefits. The techniques for tuning database checkpoint activity fall into two categories:

- Make it bigger
- Adjust checkpoint-related `init.ora` parameters

### Oracle Objective

| Monitor and tune checkpoints |

### Make it Bigger

One way to reduce the number of checkpoint events in the database is to increase the size of the RedoLogs. Since checkpoints occur at every log switch, making them larger will reduce how frequently the logs fill with redo entries and switch. Since RedoLogs cannot be altered to a larger size, the only way to increase the size of the database’s existing RedoLogs is to add new logs with the new, larger size, and then drop the original, smaller logs.

The number of redo blocks between the current checkpoint position in the RedoLog and the end of the RedoLog will not exceed 90 percent of the size of the RedoLog. If the number of blocks exceeds this size, an incremental checkpoint will occur.

To balance the recovery and performance implications of checkpoints, Oracle recommends sizing your RedoLogs so that they switch approximately every 20 to 30 minutes.

### Checkpoint-related `init.ora` Parameters

The purpose of the database checkpoint event is twofold: to assure that all buffers in the Database Buffer Cache that contain committed transactions are written to disk, and to limit instance recovery time by recording that fact in the headers of the datafiles and control files. Oracle9i provides several `init.ora` parameters that allow the DBA to tune both of these actions. However, Oracle recommends that only one be used in Oracle9i for tuning checkpoint activity: `FAST_START_MTTR_TARGET`. This `init.ora` parameter is used to specify a mean time (in seconds) to recover the instance following an instance failure. Valid values are zero to 3,600 seconds. By setting this
parameter, Oracle will perform additional checkpoints, beyond the ones performed at a log switch, in order to ensure that the checkpoints are happening frequently enough to meet the recovery target. This parameter is superseded by the FAST_START_IO_TARGET and LOG_CHECKPOINT_INTERVAL parameters if they are configured. However, both of these parameters are backward compatible Oracle8i parameters that are now deprecated in favor of the Oracle9i FAST_START_MTTR_TARGET parameter. The FAST_START_MTTR_TARGET can be set dynamically using the ALTER SYSTEM command or manually in init.ora. The default value for FAST_START_MTTR_TARGET is zero, which has the effect of disabling its functionality.

The FAST_START_MTTR_TARGET parameter is useful when Service Level Agreements dictate the maximum allowable outage related to an instance failure.

Once the FAST_START_MTTR_TARGET parameter is set, you use the V$INSTANCE_RECOVERY view to monitor its effectiveness. Selected columns from the V$INSTANCE_RECOVERY view which are related to this parameter are shown in Table 7.3.

**TABLE 7.3** Descriptions of Selected Columns in V$INSTANCE_RECOVERY

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TARGET_MTTR</td>
<td>Indicates the target Mean Time to Recover. This will usually match the value specified in the init.ora, but may be slightly higher if limited system resources will not allow the specified target to be maintained.</td>
</tr>
<tr>
<td>ESTIMATED_MTTR</td>
<td>Indicates the estimated mean time to recover the instance if it were to fail at that moment.</td>
</tr>
</tbody>
</table>

Using these columns, we can determine if our objectives for instance recovery times are being maintained:

```
SQL> SELECT target_mtt, estimated_mtt
2  FROM v$instance_recovery;
TARGET_MTT ESTIMATED_MTT
----------- ---------------
218        12
```
The output from the above query shows that our estimated mean time to recover the instance is 12 seconds, well under the target mean time to recover of 218 seconds. This indicates that the frequency of our checkpoint events is properly tuned given our instance recovery needs. Similar information regarding mean time to recover is included in the STATSPACK output. A sample STATSPACK report containing this information is shown in Figure 7.10.

**FIGURE 7.10** STATSPACK output showing MTTR statistics

<table>
<thead>
<tr>
<th>Facet</th>
<th>B</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>MTTR</td>
<td>10</td>
<td>12</td>
</tr>
<tr>
<td>Recovery</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Actual</td>
<td>742</td>
<td>936</td>
</tr>
<tr>
<td>Target</td>
<td>742</td>
<td>936</td>
</tr>
<tr>
<td>Log File Size</td>
<td>10482</td>
<td>10482</td>
</tr>
<tr>
<td>Log Okpt Timeout</td>
<td>23</td>
<td>23</td>
</tr>
<tr>
<td>Log Okpt Interval</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Similar columns for monitoring the deprecated FAST_START_IO_TARGET and LOG_CHECKPOINT_INTERVAL parameters are also included in the V$INSTANCE_RECOVERY view.

**Tuning Online Redo Log Files**

All redo information is eventually written to the Online Redo Logs by LGWR. Since the loss of an Online Redo Log can result in the loss of committed transactions, these important files should be multiplexed so that each Redo Log group contains two or more members. Tuning Redo Log performance is critical to ensure not only optimal performance, but also optimal recovery.

Measuring the performance of the Redo Logs is somewhat elusive. There are not as many dynamic performance views available for measuring Redo Log performance as there are for monitoring other redo mechanisms.

**Measuring the Performance of Redo Logs**

Measuring the performance of the database Redo Logs is difficult because Oracle does not provide many data dictionary views that directly show I/O
activity against the Redo Logs. However, the V$SYSTEM_EVENT dynamic performance view and selected OS utilities can be useful when monitoring Redo Log performance.

### Oracle Objective
Monitor and tune Redo Logs

#### Using V$SYSTEM_EVENT to Measure Redo Log Performance

As with checkpoint performance, the V$SYSTEM_EVENT table can also be used to monitor the activity of the Redo Logs. Two events, `log file switch completion` and `log file parallel write`, can be good indicators of Redo Log performance.

**Log File Switch Completion**  This event indicates the amount of time that LGWR waited for a log switch to complete. High or increasing values for this event indicate that Redo Logs may be a performance bottleneck.

**Log File Parallel Write**  This event indicates how long it took for LGWR to write the redo entries from the Redo Log Buffer to the Online Redo Logs. High or increasing values for this event can indicate that the Redo Logs are a performance bottleneck.

Figure 7.11 shows a sample query on V$SYSTEM_EVENT, checking for occurrences of these two events.

#### Figure 7.11  Redo Log activity shown in V$SYSTEM_EVENT

```sql
SQL> SELECT event, total_waits, average_wait FROM v$system_event
  2  WHERE event = 'log file switch completion'
  3  OR event = 'log file parallel write';

<table>
<thead>
<tr>
<th>EVENT</th>
<th>TOTAL_WAITS</th>
<th>AVERAGE_WAIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>log file parallel write</td>
<td>71041</td>
<td>0</td>
</tr>
<tr>
<td>log file switch completion</td>
<td>57</td>
<td>38</td>
</tr>
</tbody>
</table>
```

The output from this query shows that 57 waits for log files to complete their switching have occurred since instance startup. The average time spent waiting for those log switches to complete is 38 hundredths of a second. The query also shows that 71,041 waits have occurred for LGWR to write.
the redo entries from the Redo Log Buffer. However, the average wait for this event was so short that it appears as an average wait of zero.

Using OS Utilities to Measure Redo Log Performance

Another way to gather I/O information about the Redo Logs is to use the V$LOGFILE view to determine the locations of the Redo Log files and then use an OS utility to monitor I/O on those devices. The Unix utilities sar and iostat are both useful for monitoring disk I/O on the devices where Redo Logs are located. A brief description of each of these Unix utilities is shown in Table 7.4.

<table>
<thead>
<tr>
<th>Utility Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sar</td>
<td>System Activity Reporter utility. This utility is used to obtain detailed information about the I/O activities against the devices where the Redo Logs are located.</td>
</tr>
<tr>
<td>iostat</td>
<td>I/O Statistics utility. This utility is also used to monitor I/O activity against the devices where the Redo Logs are located. However, the output from iostat is less verbose than that of the sar utility.</td>
</tr>
</tbody>
</table>

The output and options for sar and iostat vary by operating system. For Unix systems, see your Unix man pages for complete details on their usage in your environment.

Performance Monitor can be used for similar monitoring on Windows 2000 systems, but the output is not as detailed.

Improving Redo Log Performance

If the above measures indicate Redo Log performance problems, the usual technique for improving their performance is to locate the Redo Logs on
the available storage devices in such a way as to minimize contention and maximum throughput.

---

**Oracle Objective**

**Monitor and tune Redo Logs**

---

**Minimizing Redo Log Contention**

One of the easiest ways to ensure that the Redo Logs will not cause any I/O performance issues is to separate them from other database files, like Datafiles, Control Files, and Archive Logs, onto their own devices. For example, separating Redo Logs from Datafiles generally helps performance since the I/O activity of the former is typically sequential and the I/O activity of the latter is random. Separating these disparate I/O patterns can help devices more efficiently perform their I/O activities.

**Maximizing Redo Log Throughput**

When the Redo Logs are separated from the other database files as described above, care should also be taken to place them on the fastest devices available. Placing the Redo Logs on fast devices will allow LGWR to perform its writes as quickly as possible. For this reason, it is not recommended that Redo Logs be placed on RAID volumes where write activity incurs additional I/O overhead.

---

**Oracle Objective**

Redo Logs can be created on raw devices in order to maximize throughput.

---

**Tuning Archiving Operations**

The process of archiving refers to the copying of the Redo Log files’ contents to a secondary destination so that the archived logs can be used for database recovery in the event of media failure or user error. While this mechanism is an important part of the backup and recovery aspect of administering a database, it can also be a significant source of I/O performance problems and should therefore be monitored and tuned accordingly.
Measuring the Performance of Archiving

A primary bottleneck for the performance of the Redo Log Buffer is the frequency and efficiency with which the archive background process, ARC0, copies the contents of the Redo Log files to the archive destination when the database is operating in archive log mode. If LGWR needs to write to a Redo Log that is currently being archived by ARC0, LGWR will wait until ARC0 finishes copying the Redo Log’s contents to the archive destination. To reduce the likelihood of this occurrence, LGWR will automatically start additional archive background processes (ARC1, ARC2, etc.) when demands on the system warrant them. However, two areas related to the archiving process can also cause performance problems. Common performance problems issues related to the archiving process are:

- The archive destination fills with archived logs, causing database processing to halt until additional space is made available.
- The ARC0 background process does not keep up with the Redo Log switching activity, causing a wait to occur when the Redo Logs wrap before ARC0 has completed archiving the contents of the log.

These events can be monitored using the V$ARCHIVE_DEST, V$ARCHIVE_PROCESSES, and V$SYSTEM_EVENT dynamic data dictionary views.

Filling the Archive Location

The location of the archived Redo Log files is specified using the init.ora parameter LOG_ARCHIVE_DEST or LOG_ARCHIVE_DEST_n where n is a value between 1 and 10, which allows you to specify up to ten archive destinations for extra assurance of recoverability. When the archive destination becomes full, database processing halts, and an error message is written to the database Alert log. Figure 7.12 shows an example of what this error message looks like.

When this error occurs, the database will remain halted until space is made available in the archive location. You can also temporarily change the archive location using the following command:

```sql
SQL> ALTER SYSTEM ARCHIVE LOG ALL TO
2  '/u04/oradata/PROD';
```

Once space is made available in the archive destination, archiving will resume, and a message will be written to the Alert log to indicate that archiving has resumed.
Chapter 7 • Tuning Redo Mechanisms

FIGURE 7.12 Alert log error message when archive destination is full

Mon Feb 18 21:54:18 2002
replication_dependency_tracking turned off {no async multistream replication}
Completed: alter database open
Mon Feb 18 21:54:17 2002
ARC0: Error 19504 Creating archive log file to \E:ORADATA\PROD\PROD_005.ARC'
ARC0: Archiving not possible; error count exceeded
ARC0: Failed to archive log 2 thread 0 log sequence 905
ARC0: Archive1 stopped, error occurred. Will continue retrying
ARC0:

ARC0 Processes Causing Waits at Log Switch

If the Redo Logs are filling and switching too frequently, LGWR may wrap around and try to write to a Redo Log that has not yet been archived by ARC0. Since a database in archive log mode cannot reuse a Redo Log until it has been archived, LGWR will hang for a moment until the Redo Log has been archived and becomes available. To prevent this, LGWR automatically starts new Archive processes (i.e., ARC1, ARC2, etc.) any time it detects that the current number of Archiver processes is insufficient to keep up with the switching of the Redo Logs. The DBA can also start additional Archiver processes at instance startup by setting the init.ora parameter LOG_ARCHIVE_MAX_PROCESSES to a value between 2 and 10 (1 is the default).

You can actively monitor the need for additional Archiver processes by querying the V$ARCHIVE_PROCESSES dynamic performance view. This view shows the status of each Archiver process and whether it is currently busy or idle.

Additionally, if the log file switch (archiving needed) event appears in the V$SYSTEM_EVENT dynamic performance view, LGWR may be held up by the actions of ARC0. This event indicates how often the writing of the contents of the Redo Log Buffer to the Online Redo Log by LGWR was interrupted by a delay for the ARC0 Oracle background process to copy the contents of the Redo Log to the archive destination.

Improving the Performance of Archiving

If you have identified the archiving mechanisms or ARC0 itself as a performance bottleneck, you can improve the performance of these processes by adding more Redo Logs, making the existing Redo Logs bigger, actively
managing the archive destinations, and creating additional ARC0 processes to assist with the archiving process.

**Add More**

Every Oracle database must have at least two Redo Log groups. However, by adding additional Redo Log groups to your database, you decrease the likelihood that LGWR will wrap around through all the Redo Logs and back to the one that may still be archiving. There is no magic number of Redo Log groups that is correct for all situations, but between 5 and 20 logs is generally adequate on most online transaction processing systems.

**Note**

In addition to having an adequate number of Redo Log groups, each group should have at least two members. Having multiple Redo Log members in each group minimizes possible data loss due to the corruption or loss of an Online Redo Log.

**Make It Bigger**

In addition to adjusting the quantity of Redo Logs, increasing their size can also cut down on the frequency of log switches and the archiving overhead that comes with them. For the purposes of checkpointing and recoverability, Oracle recommends that a log switch take place approximately every 20 to 30 minutes. However, as we have seen, checkpoint frequency can be tuned using the `FAST_START_MTTR_TARGET` parameters in `init.ora`. Therefore, it is possible to increase the size of your Redo Logs without negatively affecting checkpoint frequency.

**Tune Archive Destinations**

There are several `init.ora` parameters that you can use to lessen the likelihood of a database stoppage due to a filled archive location. These parameters allow you to specify multiple, concurrent archive destinations so that database transaction processing can continue even if one of the other locations fills with archive logs.

**LOG_ARCHIVE_DUPLEX_DEST** This parameter is used in conjunction with the `LOG_ARCHIVE_DEST` parameter discussed in the previous section. By setting this parameter, ARC0 is instructed to write archived Redo Logs
to both locations. When used in conjunction with LOG_ARCHIVE_MIN_SUCCEED_DEST shown in the next section, ARC0 can continue to archive Redo Log files even if one of the two locations is full.

**LOG_ARCHIVE_DEST_n** By setting the LOG_ARCHIVE_DEST_n parameter, ARC0 is instructed to write archived Redo Logs to multiple locations. These locations can be local or remote and can be specified as either required or optional when used in conjunction with LOG_ARCHIVE_MIN_SUCCEED_DEST and LOG_ARCHIVE_DEST_STATE_n. As shown in the next sections, ARC0 can continue to archive Redo Log files even if one of the locations is full.

The init.ora parameter LOG_ARCHIVE_DEST_n is only available in the Oracle9i Enterprise Edition. LOG_ARCHIVE_DEST and LOG_ARCHIVE_DUPLEX_DEST must be used to specify multiple archive locations in the Oracle9i Standard Edition.

You cannot use both the LOG_ARCHIVE_DUPLEX_DEST and LOG_ARCHIVE_DEST_n parameters at the same time. They are mutually exclusive.

**LOG_ARCHIVE_MIN_SUCCEED_DEST** When used with LOG_ARCHIVE_DUPLEX_DEST and LOG_ARCHIVE_DEST_n, this parameter specifies the number of locations to which ARC0 must be able to successfully write archive log files before an error is raised. The default value is 1. Valid values are 1 or 2 when used with LOG_ARCHIVE_DUPLEX_DEST, or between 1 and 5 when used with LOG_ARCHIVE_DEST_n.

**LOG_ARCHIVE_DEST_STATE_n** This init.ora parameter is used in conjunction with LOG_ARCHIVE_DEST_n. A LOG_ARCHIVE_DEST_STATE_n is specified for each corresponding LOG_ARCHIVE_DEST_n parameter. The valid values for this parameter are ENABLE or DEFER. The following sample from an init.ora shows how these parameters might be specified on a UNIX system running Oracle9i Enterprise Edition:

```
log_archive_dest_1 = /u01/oradata/PROD
log_archive_dest_2 = /u02/oradata/PROD
log_archive_dest_3 = /u03/oradata/PROD
```
log_archive_dest_state_1 = enable
log_archive_dest_state_2 = defer
log_archive_dest_state_3 = enable

Using these parameters, ARC0 would copy the contents of the Redo Logs to two locations: /u01/oradata/PROD and /u03/oradata/PROD. Although the location /u02/oradata/PROD is configured as an archive location, it would not be used until its status is changed from DEFER to ENABLE. Changing a location’s status from DEFER to ENABLE can be accomplished by either editing the init.ora, or by issuing the ALTER SYSTEM SET LOG_ARCHIVE_DEST_STATE_n = ENABLE command.

The dynamic data dictionary views V$ARCHIVE_DEST and V$ARCHIVED_LOG can be used to monitor the location and status of the archive activity in each archive destination.

You can further reduce the likelihood that the Archiver process will create a bottleneck by specifying a device with fast I/O capabilities for the location for your archived Redo Log files.

Raw devices cannot be used to store archive logs.

Create Additional ARC0 Processes

LGWR implicitly starts additional Archiver processes any time the number of processes is inadequate to meet the instance’s archiving needs. However, you may want to experiment with explicitly starting additional Archiver processes if ARC0 appears to be causing a performance bottleneck. Additional Archiver processes are started by using the LOG_ARCHIVE_MAX_PROCESSES parameter in the init.ora. The number of Archiver processes to start is somewhat dependent upon the number of archive destinations you have specified and the speed of the devices that are behind those locations. The default value for this parameter is 1. The valid values for this parameter are from 1 to 10. These Archive processes will start immediately at instance startup.
The number of Archiver processes is automatically set to four whenever the init.ora parameter DBWR_IO_SLAVES is set to a non-zero value. See Chapter 8, “Tuning Disk I/O,” for more details on Database Writer I/O slaves.

The dynamic data dictionary view V$ARCHIVE_PROCESSES can be used to monitor the number and activity of multiple Archiver processes. Descriptions of the contents of the V$ARCHIVE_PROCESSES view are shown in Table 7.5.

**TABLE 7.5** Contents of the V$ARCHIVE_PROCESSES View

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROCESS</td>
<td>Identifies which of the 10 Archiver processes the statistics are associated with (e.g., ARC0, ARC1, ARC2).</td>
</tr>
<tr>
<td>STATUS</td>
<td>Current status of the associated Archiver process. Valid values are STOPPED, SCHEDULED, STARTING, ACTIVE, STOPPING, TERMINATED.</td>
</tr>
<tr>
<td>LOG_SEQUENCE</td>
<td>The log sequence number of the Redo Log that is currently being archived.</td>
</tr>
<tr>
<td>STAT</td>
<td>The current state of the associated Archiver process, either IDLE or BUSY.</td>
</tr>
</tbody>
</table>

The query shown in Figure 7.13 shows a sample query on V$ARCHIVE_PROCESSES.

**FIGURE 7.13** Sample query on V$ARCHIVE_PROCESSES

```
SQL> SELECT process, status, log_sequence, stat
2   FROM v$archive_processes;

<table>
<thead>
<tr>
<th>PROCESS</th>
<th>STATUS</th>
<th>LOG_SEQUENCE</th>
<th>STAT</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>ACTIVE</td>
<td>1000231</td>
<td>BUSY</td>
</tr>
<tr>
<td>1</td>
<td>ACTIVE</td>
<td>1000232</td>
<td>BUSY</td>
</tr>
<tr>
<td>2</td>
<td>ACTIVE</td>
<td>1000233</td>
<td>IDLE</td>
</tr>
<tr>
<td>3</td>
<td>ACTIVE</td>
<td>1000233</td>
<td>IDLE</td>
</tr>
<tr>
<td>4</td>
<td>STOPPED</td>
<td>0</td>
<td>IDLE</td>
</tr>
<tr>
<td>5</td>
<td>STOPPED</td>
<td>0</td>
<td>IDLE</td>
</tr>
<tr>
<td>6</td>
<td>STOPPED</td>
<td>0</td>
<td>IDLE</td>
</tr>
<tr>
<td>7</td>
<td>STOPPED</td>
<td>0</td>
<td>IDLE</td>
</tr>
<tr>
<td>8</td>
<td>STOPPED</td>
<td>0</td>
<td>IDLE</td>
</tr>
<tr>
<td>9</td>
<td>STOPPED</td>
<td>0</td>
<td>IDLE</td>
</tr>
</tbody>
</table>
```
The output in Figure 7.13 shows that four Archiver processes are started and that two of them (0 and 1) are busy copying the contents of Redo Log sequence numbers 100231 and 100232 to the archive destination(s).

Oracle recommends starting additional Archiver processes if the V$ARCHIVE_PROCESSES view consistently shows that two or more logs require archiving.

Summary

This chapter has shown you the important role that Oracle’s redo mechanisms play in maintaining Oracle’s high degree of transaction recoverability. Redo information is initially placed in the Redo Log Buffer by each user’s Server Process, where it remains until LGWR writes the contents of Redo Log Buffer to the Online Redo Log files. When the Online Redo Log fills with redo entries, LGWR will switch to the next available Redo Log, causing the Archiver process to copy the contents of the Redo Log to the archive destination(s) if the database is operating in archive log mode.

Several dynamic performance views can be used to monitor the components of the redo mechanisms, including: V$SYSSTAT, V$SESSION_WAIT, V$SYSTEM_EVENT, V$INSTANCE_RECOVERY, V$LOGFILE, V$ARCHIVE_DEST, V$ARCHIVE_PROCESSES, and V$ARCHIVED_LOG. The output from STATSPACK and the REPORT.TXT file also contain redo tuning information.

Techniques for improving the performance of the Redo Log mechanisms include increasing the size of the Redo Log Buffer, reducing redo entries by using the UNRECOVERABLE and NOLOGGING options, improving the efficiency of checkpoints, adding additional Redo Logs, making the existing Redo Logs larger, and improving the performance of the Archiver Oracle background process by starting multiple Archiver processes.

Exam Essentials

Know how to monitor the performance of the Redo Log Buffer. Understand which statistics and wait events indicate potential problems with the Redo Log Buffer. Be able to describe what types of changes can be made to improve the performance of the Redo Log Buffer.
Know how to monitor the performance of the Redo Logs. Be able to describe how Oracle’s Redo Logging mechanisms are affected by changes to the size and number of Redo Logs. Know which dynamic performance views can be used to monitor Redo Log performance.

Understand the importance of checkpoints. Know when checkpoint events occur and what happens during a checkpoint event. Be able to describe where information about checkpoint activity can be found and which init.ora parameters can be used to tune checkpoint activity.

Key Terms

Before you take the exam, make sure you are familiar with the following terms:

- archived Redo Log
- Archiver
- database checkpoint
- Log Writer
- Online Redo Log
- Redo Log Buffer
Review Questions

1. Which of the following processes helps manage the contents of the Redo Log Buffer?
   A. LRU List
   B. Dirty List
   C. LGWR
   D. All of the above play a role in managing the Redo Log Buffer.

2. Which of the following command types do not place entries in the Redo Log Buffer?
   A. INSERT
   B. SELECT
   C. UPDATE
   D. DROP TABLE

3. Which event represents the moment in time when the database is in a consistent state?
   A. Redo Log write
   B. Database write
   C. Checkpoint
   D. Cache flush

4. Which of the following events will not cause LGWR to write the entire contents of the Redo Log Buffer to disk?
   A. The application user issues a COMMIT.
   B. The database is shutdown normally.
   C. A checkpoint occurs.
   D. The application user issues an UPDATE.
5. Which of the following actions cause LGWR to empty the contents of the Redo Log Buffer to the Online Redo Log? (Choose two.)
   A. Occurrence of a database checkpoint event.
   B. ARC0 copies a Redo Log to the archive destination.
   C. Any user issues a COMMIT.
   D. DBW0 writes to a datafile.

6. You want to make a copy of a large table. What keyword could be added to the ‘CREATE TABLE AS’ syntax to minimize the impact on the Redo Logs?
   A. NOLOG
   B. NOLOGGING
   C. NOCACHE
   D. NORECOVER

7. If the EMP table is specified as LOGGING, which of the following operations will generate redo information for rows inserted into the table?
   A. The INSERT DML statement.
   B. Direct Path loads using SQL*Loader.
   C. Direct load inserts using the /*+ APPEND */ hint.
   D. All of the above cause redo information to be generated.

8. All of the following will cause a database checkpoint except:
   A. DBA performs a SHUTDOWN TRANSACTIONAL.
   B. An application user issues a COMMIT.
   C. Redo log switches.
   D. Any of the above can cause a database checkpoint event to occur.

9. Which dynamic performance view can be used to calculate the Redo Log Buffer Retry Ratio?
A. V$SYSSTAT
B. V$SESSION
C. V$SYSTEM_EVENT
D. V$SGASTAT

10. Which statistic in the V$SESSION_WAIT dynamic performance view can be used to determine which users are currently experiencing waits for access to the Redo Log Buffer?
   
   A. buffer wait
   B. redo log buffer wait
   C. log buffer space
   D. log buffer wait

11. Which statistics in the output from a REPORT.TXT file can be used to identify excessive checkpoint activity?
   
   A. check point start/checkpoint stop
   B. database checkpoints started/database checkpoint ended
   C. background checkpoints started/background checkpoints completed
   D. background checkpoint start time/background checkpoint end time

12. You look in the init.ora file of your production instance and see the following line: FAST_START_MTTR_TARGET = 240. Which answer best explains what this line means?
   
   A. A checkpoint will be performed every 240 seconds
   B. Oracle will perform no more than 240 checkpoints per day
   C. Oracle will perform sufficient checkpoints to ensure that the mean time to recover after instance failure will not exceed 240 seconds.
   D. Oracle will ensure that it will not have to apply more than 240 checkpoints to recover from an instance failure.
13. According to Oracle, in a well-tuned database, user Server Processes should find space in the Redo Log Buffer without waiting approximately:
   A. One percent of the time
   B. Ninety-nine percent of the time
   C. Fifty-five percent of the time
   D. Eighty percent of the time

14. Which of the following methods can be used to improve the performance of the Redo Log Buffer?
   A. Increase the value for the LOG_BUFFER init.ora parameter.
   B. Reduce the amount of redo generated by using the NOLOGGING option.
   C. Avoid unnecessary checkpoints.
   D. All of the above are effective methods of improving the performance of the Redo Log Buffer.

15. Creating a table with the NOLOGGING option will cause no redo entries to be placed in the Redo Log Buffer except when the following types of DML are performed on the table:
   A. Direct path SQL*Loader loads
   B. Regular DML
   C. Inserts done with the /*+ APPEND */ hint
   D. Inserts done as a result of a CREATE TABLE AS SELECT subquery.

16. Which dynamic performance view can be queried to gather information on the performance of the Online Redo Logs?
   A. V$SYSTEM_EVENT
   B. V$REDO_LOG
   C. V$LOGFILE
   D. V$SYSTEM_STATS
17. The size of the Redo Log Buffer is specified by which of the following init.ora parameters?
   A. LOG_BUFFER_SIZE
   B. REDO_LOG_BUFFER
   C. REDO_LOG_BUFFERS
   D. LOG_BUFFER

18. You increase the size of the Redo Logs from 20MB to 50MB. If the init.ora parameter FAST_START_MTTR_TARGET is set to zero, what effect will this increase in Redo Log size have on checkpointing?
   A. Checkpointing will occur more frequently.
   B. Checkpointing will occur less frequently.
   C. Checkpointing will not be affected.

19. LGWR starts to write the contents of the Redo Log Buffer to disk when it reaches what point?
   A. Two-thirds full
   B. One-third full
   C. One-half full
   D. One hundred percent full

20. What are the two possible values of the LOG_ARCHIVE_DEST_STATE_n parameter? (Choose two.)
   A. START
   B. DISABLED
   C. DEFER
   D. ENABLE
Answers to Review Questions

1. C. The LGWR background process writes the contents of the Redo Log Buffer to the Online Redo Log to ensure that user Server Processes can find room for their entries in the buffer.

2. B. Only DML and DDL statements require entries in the Redo Log Buffer.

3. C. Checkpoint events mark a point in time when the database was made consistent. Modified database buffers in the buffer cache are flushed to disk. The significance of this event is that if an instance failure should occur, Oracle only needs to recover transactions from the last completed checkpoint event to the time of the failure, hence expediting the recovery process.

4. D. DML statements like INSERT, UPDATE, or DELETE will not trigger a checkpoint to occur.

5. A, C. LGWR also writes every three seconds, whenever the Redo Log Buffer is one-third full, or when 1MB of redo information has been written to the Redo Log Buffer.

6. B. The NOLOGGING keyword will cause Oracle to not log the insertion of rows into the new table from the old table. This will greatly minimize the impact on the Redo Logs.

7. D. By specifying a table as LOGGING, you are requesting that all DML actions against the table be logged. This is the default behavior for Oracle.

8. B. A user COMMIT causes LGWR to empty the contents of the Redo Log Buffer to the Online Redo Log, but does not cause a database checkpoint.

9. A. The V$SYSSTAT dynamic performance view contains the redo entries and redo buffer allocation retries statistics that can be used to calculate how often user Server Processes are waiting to place entries in the Redo Log Buffer.

10. C. The statistic log buffer space indicates which users are currently waiting, or have recently waited, for access to the Redo Log Buffer.
11. C. If the number of background checkpoints started and number of background checkpoints completed differ by more than 1, checkpoints are not always completing before the next one begins.

12. C. This parameter specifies the desired mean time to recover an instance. From this measure, Oracle will then perform checkpointing sufficient to accommodate this requirement.

13. B. The ratio of Redo Log Buffer entries to Redo Log Buffer retries should be less than 1 percent.

14. D. Another option for improving the performance of the Redo Log Buffer is to improve the speed of Redo Log archiving.

15. B. Regular application DML is still recorded in the Redo Log Buffer even if a table is a NOLOGGING table.

16. A. The V$EVENT view contains information about Online Redo Log performance. The statistics to take note of are log file switch complete and log file parallel write.

17. D. The size of the Redo Log Buffer is specified in bytes using the LOG_BUFFER parameter in init.ora.

18. B. Because checkpoints occur on log switches, increasing the size of the Redo Logs will decrease the frequency of checkpoints. This is one technique to manage the checkpoint frequency.

19. B. The LGWR background process writes the contents of the Redo Log Buffer to the Online Redo Log when the buffer is one-third full. This helps ensure that space will always be available for subsequent transactions.

20. C, D. The two valid values are ENABLE and DEFER. When set to ENABLE, the Archiver process will write archive logs to the corresponding LOG_ARCHIVE_DEST_n destination. When set to DEFER, the Archiver process will not write archive logs to the corresponding LOG_ARCHIVE_DEST_n destination until the value is changed to ENABLE.
Tuning Disk I/O

ORACLE9i PERFORMANCE TUNING
EXAM OBJECTIVES COVERED IN THIS CHAPTER:

✓ Explain the advantages of distributing different Oracle file types
✓ Describe reasons for partitioning data in tablespaces
✓ Diagnose tablespace usage problems
✓ Describe the correct usage of extents and Oracle blocks
✓ Describe the use of Oracle block parameters
✓ Explain space usage and the High Water Mark
✓ Determine the High Water Mark
✓ Recover space from sparsely populated segments
✓ Describe and detect chaining and migration of Oracle blocks
✓ Describe how sorts are performed
✓ Differentiate between disk and memory sorts
✓ Identify SQL operations which require sorts
✓ Create and monitor temporary tablespaces
✓ Reduce total sorts and disk sorts
✓ Determine the number of sorts performed in memory
✓ Set old and new sort parameters
✓ Use the dynamic performance views to check rollback segment performance
✓ Reconfigure and monitor rollback segments
✓ Define the number and sizes of rollback segments
✓ Appropriately allocate rollback segments to transactions
✓ Explain the concept of automatic undo management
✓ Create and maintain automatic managed undo tablespace
✓ Configure the instance to use I/O slaves
✓ Configure and use multiple DBW processors

Exam objectives are subject to change at any time without prior notice and at Oracle’s sole discretion. Please visit Oracle’s Certification website (http://www.oracle.com/education/certification/) for the current exam objectives listing.
Nearly every action that occurs in the database will result in some type of I/O activity. That I/O activity can either be logical (in memory) or physical (on disk). Chapters 4, 5, 6, and 7 all discussed techniques for minimizing physical I/Os by tuning the components of the SGA so that most I/Os would be logical reads instead of physical ones. This is very important from a performance perspective because logical I/Os are thousands of times faster than physical I/Os. However, no degree of memory tuning will completely eliminate physical I/O activity. For example, all of the following activities can be the source of disk I/Os:

- Database Writer (DBW0) writing data buffers from the Database Buffer Cache to the database’s Datafiles
- DBW0 writing data to rollback segment blocks to maintain read consistency
- User Server Processes reading data blocks into the Database Buffer Cache
- Log Writer (LGWR) writing transaction recovery information from the Redo Log Buffer to the online redo log
- Archiver (ARC0) reading the contents of redo logs and writing those contents to the archive destination
- Application user activity temporarily writing large sort requests to disk

Any of these actions can cause performance difficulties if they occur excessively or are done inefficiently. Therefore, the goals when tuning physical I/O are generally these:

- Minimize physical I/O whenever possible by properly sizing the SGA.
- Perform any remaining physical I/O as fast as possible when it is required.
Therefore, this chapter addresses the important area of tuning database disk I/O. The following sections will discuss how to measure and tune the disk I/O performance of these database structures:

- Datafiles
- DBW0
- Individual segment data blocks
- Sort activity and temporary segments
- Rollback segments

These measures will be accomplished using queries on V$ and DBA views, examining the contents of STATSPACK and REPORT.TXT reports, and the OEM Performance Manager GUI tool. The first section discusses tuning Datafile I/O.

Other important I/O tuning considerations related to Redo Logs, checkpoint activity, and archiving were discussed in Chapter 7, “Tuning Redo Mechanisms.”

Tuning Tablespace and Datafile I/O

Every database segment is stored within a logical structure called a Tablespace. Every database has at least one tablespace, SYSTEM, which is used to store the Oracle data dictionary tables and indexes. Well-designed databases will also have many additional tablespaces to hold non–data dictionary, application segments. The “correct” number of tablespaces to have in a database varies depending the type of application, number and type of application segments, number of application users, and the type of transactions the application users are performing. For example, transaction processing databases created using Oracle’s GUI database creation tool, Database Configuration Assistant, will have the tablespaces shown in Table 8.1.

Although the tablespaces shown in Table 8.1 are only one possible way to configure a database, they do demonstrate an important concept. A properly designed and well-tuned database will make use of many different tablespaces so that dissimilar segments (e.g., application tables and application indexes) are each stored in their own separate tablespaces.
In addition, having several tablespaces, segregated by function, makes database management and backup activities much easier. For example, putting rollback segments in their own tablespace will help ensure that user transactions will always find adequate space to store their before-image data. Additionally, tablespaces that contain only temporary segments do not need to be backed up since they will not store any application data. In general, you should have at least six tablespaces in your database:

1. A SYSTEM tablespace for data dictionary segments. This tablespace is required in every database. No application segments should be stored in the SYSTEM tablespace.

2. A tablespace that stores only rollback or undo segments. A detailed discussion of techniques for managing rollback and undo segments is included later in this chapter.

<table>
<thead>
<tr>
<th>Tablespace Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CWMLITE</td>
<td>Stores segments for Online Analytical Processing (OLAP) processing</td>
</tr>
<tr>
<td>DRSYS</td>
<td>Stores segments needed for Oracle’s Context feature</td>
</tr>
<tr>
<td>EXAMPLE</td>
<td>Stores the segments for the sample schemas (e.g., HR, OE)</td>
</tr>
<tr>
<td>INDX</td>
<td>Stores your application indexes</td>
</tr>
<tr>
<td>SYSTEM</td>
<td>Stores data dictionary segments</td>
</tr>
<tr>
<td>TEMP</td>
<td>Temporary tablespace used for sorting</td>
</tr>
<tr>
<td>TOOLS</td>
<td>Stores non-application segments for utilities</td>
</tr>
<tr>
<td>UNDOTBS</td>
<td>Stores rollback segments</td>
</tr>
<tr>
<td>USERS</td>
<td>Stores segments created by users</td>
</tr>
</tbody>
</table>

**TABLE 8.1** Database Configuration Assistant Default Tablespaces
3. A tablespace that stores temporary segments created during user sort activities. Every application user should have this tablespace specified as their temporary tablespace. Tuning sort activities is discussed in detail later in this chapter.

4. A tablespace for application tables. You may want several tablespaces for application tables. For example, one way to further partition your application’s tablespaces might be to create tablespaces for small tables, medium tables, and large tables. Separate tablespaces that contain read-only data can also be useful. Additional topics related to table storage, like extent and block management, are discussed later in this chapter.

5. A tablespace for application indexes. You may want several tablespaces for application indexes. In addition to partitioning by size, you could also consider partitioning your application’s index tablespaces by application module—creating separate tablespaces for HR (human resources) tables, OE (order entry) tables, and SH (shipping) tables.

6. A tablespace to store data for large objects and the overflow segments associated with Index-Organized Tables (IOTs).

---

**NOTE**

You may also want to create a tablespace for non-application schema that need to store segments in the database. Examples of these types of schema include the Enterprise Manager and Recovery Manager repository tables.

No matter how many tablespaces you have in your database, each tablespace will comprise one or more physical files called **Datafiles**. The actual I/O occurs against the Datafiles, not the tablespaces themselves. Because of this, the number and location of the physical Datafiles that make up your database’s tablespaces are an important area of tuning consideration. In particular, you should try to minimize disk contention by placing Datafiles for the various tablespaces on different disk drives, volume groups, or disk controllers in order to maximize I/O performance. The next section discusses how to measure Datafile I/O activity.

---

**TIP**

Not only is it important to tune Datafile placement in relation to other Datafiles, care should also be taken to separate Datafiles, Redo Logs, and Archive Logs. This concept is discussed in greater detail in the section, “Balancing Datafile I/O to Improve Performance.”
Measuring Datafile I/O

Measuring the physical I/O performance of the Datafiles in the database can be done using the V$FILESTAT and V$DATAFILE dynamic performance views, the output from STATSPACK and REPORT.TXT, and the output from the OEM Performance Manager GUI tool.

Using V$FILESTAT, V$DATAFILE, and V$TEMPFILE to Measure Datafile I/O

The V$FILESTAT, V$DATAFILE, and V$TEMPFILE dynamic performance views can be used to monitor the performance of the read and write activity against the individual Datafiles and tablespaces in the database. A description of selected columns from these three views appears in Tables 8.2, 8.3, and 8.4.

TABLE 8.2 Description of Selected V$FILESTAT Columns

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FILE#</td>
<td>The Datafile's internal identifier assigned by Oracle at tablespace creation</td>
</tr>
<tr>
<td>PHYRDS</td>
<td>Number of physical reads done on that Datafile</td>
</tr>
<tr>
<td>PHYWRTS</td>
<td>Number of physical writes done to that Datafile</td>
</tr>
<tr>
<td>AVGIOTIM</td>
<td>Average time, in milliseconds, spent performing I/O on that Datafile</td>
</tr>
<tr>
<td>MINIOTIM</td>
<td>Minimum time, in milliseconds, spent performing I/O on that Datafile</td>
</tr>
<tr>
<td>MAXIOWTM</td>
<td>Maximum time, in milliseconds, spent writing to that Datafile</td>
</tr>
<tr>
<td>MAXIORTM</td>
<td>Maximum time, in milliseconds, spent reading from that Datafile</td>
</tr>
</tbody>
</table>
Figure 8.1 shows a query on these three views that displays the overall statistics for every Datafile (denoted by D in the first column) and tempfile (denoted by T in the first column) in the database.

Figure 8.1 Performance query on V$FILESTAT, V$DATAFILE, and V$TEMPFILE

```sql
SQL> SELECT s.file#, s.filetype, s.name "DATAFILE", d.file#, d.name "TEMPFILE",
  2  s.phyrds, s.phywrts, s.avgiotim, s.miniortim, s.maxiortim, s.maxiowtm
  3  FROM v$filestat s, v$datafile d
  4  WHERE s.file#=d.file#
  5
  6  SELECT s.file#, t.name "DATAFILE",
  7  s.phyrds, s.phywrts, s.avgiotim, s.miniortim, s.maxiortim, s.maxiowtm, s.maxiortm
  8  FROM v$filestat s, v$tempfile t
  9  WHERE s.file#=t.file#
  10  ORDER BY 1
```

The init.ora parameter TIMED_STATISTICS must be set to TRUE; otherwise, the values for the PHYRDS, PHYWRTS, AVGIO TIM, MINIOTIM, MAXIOWTM, and MAXIORTM columns will all be zeros.
For more details on tempfiles, see the section headed “Tuning Sort I/O” later in this chapter.

Using the output in Figure 8.1, you can see that the majority of physical writes are occurring against three Datafiles: appl_idx01.dbf, undo01.dbf, and appl_tab01.dbf, which are the Datafiles for the application indexes, rollback segments, and tables respectively. The majority of physical reads are occurring on the Datafile (readonly01.dbf) that makes up the read-only tablespace. As you’ll see in the next section, the placement of these files on disks relative to one another is an important tuning consideration.

The SYSTEM tablespace Datafiles will frequently experience high read activity because the data dictionary tables that Oracle uses to perform recursive SQL are located in the SYSTEM tablespace. However, excessive write activity to the SYSTEM tablespace may indicate that non–data dictionary segments are stored there.

**Using STATSPACK to Measure Datafile I/O**

The output from the STATSPACK feature contains similar information regarding Datafile and tablespace I/O characteristics. Figure 8.2 shows sample output from a STATSPACK report.

**FIGURE 8.2 STATSPACK output showing Datafile I/O statistics**
The output shown in Figure 8.2 indicates that during the period between the two STATSPACK snapshots, the TEMP tablespace received the most read and write activity. Notice that unlike the output in V$FILESTAT, the STATSPACK output expresses the duration of the read and write activity in both seconds and milliseconds. This information is useful when determining where to physically locate Datafiles on the available disk drives, volume groups, and controllers.

**Using REPORT.TXT to Measure Datafile I/O**

The REPORT.TXT file generated by UTLBSTAT.SQL and UTLSESTAT.SQL also contains file I/O information at both the tablespace level and Datafile level. Figure 8.3 shows the REPORT.TXT output that displays tablespace I/O statistics. Figure 8.4 shows the section of REPORT.TXT that contains tablespace and Datafile I/O statistics.

**FIGURE 8.3** Sample REPORT.TXT showing tablespace I/O statistics

**FIGURE 8.4** Sample REPORT.TXT showing tablespace and Datafile I/O statistics
In Figure 8.3, you can see that the majority of the read and write activity I/O took place on the APPL_IDX, USERS, and SYSTEM tablespaces. You can also see that in some cases, multiple blocks were read and written with a single I/O. For example, the USERS tablespace had 648 physical reads performed against it, reading 3,755 blocks. This is approximately five to six database blocks read for every physical read on that tablespace. Tuning this type of multiblock read is discussed in the Datafile and tablespace tuning section later in this chapter. Figure 8.4 shows that I/O on the APPL_TAB tablespace was distributed across the tablespace’s two Datafiles, APPL_TAB01.DBF and APPL_TAB02.DBF. However, approximately 70 percent of the I/O was on the APPL_TAB01.DBF Datafile. This information can also be useful when tuning Datafile disk I/O.

There is no standard rule as to which Datafiles should be receiving the most or least I/O. In general, however, you should strive to balance the I/O requests across several Datafiles.

**Using Performance Manager to Measure Datafile I/O**

The Performance Manager component of the Oracle Enterprise Manager Diagnostics Pack includes several graphical representations of Datafile I/O performance. Figure 8.5 shows a sample Performance Manager screen that displays I/O activity based on time for each Datafile in the database.
The output shown in Figure 8.5 indicates that the SYSTEM01.DBF and USERS01.DBF spent the most time performing database I/O during the OEM monitoring period.

**Improving Datafile I/O**

Once statistics about Datafile I/O patterns have been gathered, there are several techniques available to the DBA to enhance the performance of each disk I/O related to Datafiles. In general, these techniques attempt to make sure that all I/Os against Datafiles are optimized by:

- Making sure that non-Oracle I/O does not interfere with Datafile I/O
- Using Locally Managed Tablespaces to reduce I/O
- Balancing Datafile I/O across many devices, logical volumes and controllers
- Performing all Datafile I/O activity as quickly as possible when it does occur

Each of these techniques is described in greater detail in each of the following sections.

---

**Oracle Objectives**

- Explain the advantages of distributing different Oracle file types
- Describe reasons for partitioning data in tablespaces

---

**Minimizing Non-Oracle I/O to Improve Performance**

In order to make your Datafile tuning efforts as effective as possible, it is important that you do not place non-Oracle files on the same disk drives or logical volumes as the Datafiles. Not only will the placement of non-Oracle files on the same disk as the Datafiles cause potential contention for those disk resources, but also any I/O against the non-Oracle files will not be included in the Datafile I/O statistics mentioned in the previous section. Therefore, these statistics will not reflect the true I/O characteristics of the Datafiles and their associated devices. This may lead to misguided tuning decisions.
Using Locally Managed Tablespaces to Improve Performance

Another technique for improving tablespace I/O is to create Locally Managed Tablespaces to store application data. A Locally Managed Tablespace (LMT) uses a bitmap stored in the header of each of the tablespace’s Datafiles instead of using Free Lists in the data dictionary to manage the allocation of space within the tablespace. This allows LMTs to allocate and de-allocate space from the tablespace more quickly, without having to access the data dictionary in the SYSTEM tablespace. Segments stored in LMTs also have the added advantage of being able to have thousands of extents without incurring any performance degradation.

Balancing Datafile I/O to Improve Performance

The simplest way to balance the I/O between the segments in the database is to allocate the storage for those segments to the appropriate tablespaces. At a minimum, this would entail separating the major components of the database into their own tablespaces and making sure that database users do not have the SYSTEM tablespace assigned as their default tablespace or temporary tablespace. Excessive I/O on the SYSTEM tablespace Datafiles can indicate that non-data dictionary segments may be stored in the SYSTEM tablespace.

Table 8.5 shows a minimum tablespace configuration that could be used to achieve these goals.

<table>
<thead>
<tr>
<th>Tablespace Name</th>
<th>Type and Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSTEM</td>
<td>Objects owned by the user SYS (i.e., the data dictionary).</td>
</tr>
<tr>
<td>TOOLS</td>
<td>Default tablespace for the user SYSTEM and any other non-application schemas.</td>
</tr>
<tr>
<td>USERS</td>
<td>Default tablespace for all other database users.</td>
</tr>
<tr>
<td>TEMP</td>
<td>Location for temporary segments used for sorting. This tablespace should use a tempfile for its associated Datafile.</td>
</tr>
</tbody>
</table>
Applications that make use of stored PL/SQL procedures, functions, triggers, and packages must still store that PL/SQL source code in the data dictionary in the SYSTEM tablespace.

### Real World Scenario

**Putting Everything in Its Proper Place**

I have worked with several packaged applications that, as delivered from the vendor, placed all their application tables and indexes in the SYSTEM tablespace. We have already seen that this is not the desired configuration from a performance and management perspective. Here are some techniques for moving these segments to their own tablespace:

- Create two new locally managed tablespaces, one to hold the application tables and one to hold the application indexes (e.g., `APPL_TAB` and `APPL_IDX`).

- Move all the application tables and indexes to the new tablespace using the `ALTER TABLE ... MOVE` and `ALTER INDEX ... REBUILD` commands respectively. You can use the following SQL script to create a second SQL script that will contain all the commands for moving the application tables to the new `APPL_TAB` tablespace:

```sql
set echo off
set feedback off
```
set pagesize 0
set verify off

PROMPT Enter the name of the application owner:
ACCEPT app_owner

PROMPT Enter the name of the new tablespace for the application tables:
ACCEPT new_tab_tablspace

PROMPT Enter the name of the new tablespace for the application indexes:
ACCEPT new_idx_tablspace

spool MoveTablesIndexes.tmp

-- move the tables to the new tablespace
SELECT 'ALTER TABLE ''||owner||'.'||table_name||CHR(10)||
     ' MOVE TABLESPACE &new_tab_tablspace;' FROM dba_tables WHERE owner = UPPER('&app_owner');

-- rebuild all indexes on the moved tables, even those not owned
-- by the specified user because moving the tables will set their
-- status to UNUSABLE (unless they are IOT tables)
SELECT 'ALTER INDEX ''||I.owner||'.'||I.index_name||CHR(10)||
       'REBUILD TABLESPACE ''||I.tablespace_name||'';' FROM dba_indexes I, dba_tables T WHERE I.table_name = T.table_name AND I.owner = T.owner AND T.owner = UPPER('&app_owner');

-- rebuild any other indexes owned by this user that may not be
-- on the above tables
As stated earlier, the separation of temporary segment and rollback segment tablespaces from the SYSTEM tablespace is important for both tuning and management purposes, as is separating application tables from their associated indexes. In particular, this technique helps improve performance on busy OLTP systems when tables and their associated indexes are having their blocks read while the blocks are also having rows inserted into them. This concept can even be taken a step further by segregating segments by functional area (i.e., separating Accounting tables and indexes from Inventory tables and indexes) or size (i.e., separating large tables and indexes from small tables and indexes).

It is also important to separate the non-Datafile database components, such as Redo Logs and Control Files, because their I/O characteristics are different. These characteristics are discussed below.

```sql
SELECT 'ALTER INDEX '||owner||'.'||index_name||CHR(10)||
      'REBUILD TABLESPACE &new_idx_tablespace;'||CHR(10)||
FROM dba_indexes
WHERE owner = UPPER('&app_owner');

spool off

set echo on
set feedback on
set pagesize 60

spool MoveTablesIndexes.log

@MoveTablesIndexes.tmp

spool off

* Take a good cold or hot database backup once the tables and indexes have been moved to the appropriate locations.

Using these techniques will not only improve the performance of your application, but will also increase the space available in the SYSTEM tablespace for any application PL/SQL objects.
Control Files  The Database Writer and Checkpoint background processes write to the Control Files. The Archiver background process reads from the Control Files during a database recovery. Since the I/Os on the Control Files are brief, they can be placed almost anywhere without adversely impacting performance. However, for recoverability purposes, at least one Control File should be located on a disk that doesn’t contain other database components.

Datafiles  The Database Writer background processes write to Datafiles. The user Server Processes reads Datafiles. Separating System, temporary, and rollback segment Datafiles from application table and index Datafiles will help enhance performance.

Redo Logs  Redo Logs are written to in a sequential fashion by the Log Writer background process. Redo Logs are also read by the Archiver process while the archived Redo Log is being created. Datafiles should be separated from Redo Logs not only for recovery purposes, but also because Datafile I/O is generally random whereas Redo Log I/O is sequential. For both recoverability and performance reasons, the Redo Logs and the archived Redo Logs should not be placed on the same device.

Archived Redo Logs  The archived Redo Logs are written to by the Archiver background process at a log switch. For both recoverability and performance reasons, the Redo Logs and the archived Redo Logs should not be placed on the same device.

Background and User Trace Files  Background and user trace files are written to the directories specified by the init.ora parameters BACKGROUND_DUMP_DEST and USER_DUMP_DEST. Trace files can be generated by any of the Oracle background processes and by user Server Processes. Since these I/Os tend to be very short in duration, placement of these files has a negligible impact on performance.

On modern disk arrays it is often difficult to truly separate Oracle database files on separate physical devices. This occurs because many arrays use sophisticated software to dynamically manage the location of files, and may even dynamically move files between devices based on I/O patterns. However, many of the above tuning techniques should still be considered when creating the logical layout of the array—even if the physical layout is handled by the array itself.
Once proper Datafile location has been considered, the next step to maximizing Datafile I/O is to perform the read and write activity to the Datafiles as quickly as possible.

**Performing Datafile I/O Quickly to Improve Performance**

After balancing the I/O requests by separating tables and indexes into their own tablespaces, you can try to perform the I/Os against these structures as quickly as possible. Performing Datafile I/O quickly is achieved in three ways: by placing high I/O Datafiles on separate disk drives and controllers, striping tablespaces, and tuning the `DB_FILE_MULTIBLOCK_READ_COUNT` init.ora parameter.

**Segregating Datafiles by Drive and Controller**

Improving I/O performance by separating table segments from their associated indexes was explained in the previous section. By additionally ensuring that two different devices are used to store these Datafiles and that these devices are attached to two separate disk controllers, I/O performance can be improved even more.

**Datafile Striping**

The idea of separating Datafiles onto distinct devices can be extended even further through a technique called **striping**. When a Datafile is striped, it is stored across several devices, not just one. This increases I/O performance because multiple sets of disk drive heads are brought into use when a read or write of the Datafile is required. The easiest way to stripe a Datafile is to store the Datafile on a **RAID** device. A RAID, or Redundant Array of Independent Disks, consists of several physical disks that can be managed and accessed as if they were one or more physical devices. Therefore, placing a Datafile on a RAID device implicitly causes the file to be striped across all the devices in the RAID. This technique does not require any special action on the part of the DBA.

---

**NOTE**

Two key considerations when using RAID striping are the stripe width and stripe depth. The number of devices across which the stripe is built determines stripe width. **Stripe depth** refers to the amount of data that will be written to one device before moving to the next. Ask your storage array vendor the recommended stripe width and depth configuration for their hardware.
Datafiles can also be manually striped across devices in the absence of a RAID array. Manual striping is accomplished by creating a tablespace made up of several Datafiles, where each Datafile is placed on a separate physical device. Next, a segment is created so that its extents are stored in each of the Datafiles associated with the tablespace, effectively striping the segment across the available devices. The following steps show how a SQL statement could be used to accomplish this for a table called EMP.

1. First, create a tablespace to hold the EMP table:

   ```sql
   SQL> CREATE TABLESPACE appl_data
   2       DATAFILE '/u01/oradata/PROD/appl_data01.dbf' SIZE 5M
   3       EXTENT MANAGEMENT LOCAL
   4       AUTOEXTEND ON NEXT 5M MAXSIZE 2000M;
   ```

2. Now, create the EMP table’s initial extent in the new tablespace:

   ```sql
   SQL> CREATE TABLE emp
   2       (emp_id number,
   3       last_name varchar2(20),
   4       first_name varchar2(30),
   5       dept_id number)
   6       STORAGE (INITIAL 4M NEXT 4M PCTINCREASE 0)
   7       TABLESPACE appl_data;
   ```

3. Next, add two additional Datafiles, each located on separate devices from the other Datafiles, to the tablespace where the EMP table is stored:

   ```sql
   SQL> ALTER TABLESPACE appl_data
   2       ADD DATAFILE '/u02/oradata/PROD/appl_data02.dbf' SIZE 5M;
   SQL> ALTER TABLESPACE appl_data
   2       ADD DATAFILE '/u03/oradata/PROD/appl_data03.dbf' SIZE 5M;
   ```

4. Finally, manually allocate two new extents for the EMP table using the Datafiles that were created:

   ```sql
   SQL> ALTER TABLE emp
   2       ALLOCATE EXTENT
   3       (DATAFILE
   ```
Figure 8.6 compares these two methods of striping a Datafile.

Manual striping tends to be a time consuming activity that may not yield benefits in proportion to the time spent managing it.
See Chapter 3, “Tablespaces and Datafiles,” in Oracle9i Concepts, Release 8.1.5 (Part Number A67781-01) for more details on the creation and management of locally managed tablespaces.

**DB_FILE_MULTIBLOCK_READ_COUNT init.ora Parameter**

The `DB_FILE_MULTIBLOCK_READ_COUNT` init.ora parameter determines the maximum number of database blocks that are read in one I/O operation by a user’s Server Process whenever a full table scan read operation is performed. The default value for this parameter is 8. The maximum value for this parameter varies by OS. Setting this parameter to a value greater than 8 benefits performance most when the database blocks are being accessed via full table scans. By increasing this parameter, more blocks are accessed with each I/O, thereby cutting down on the total I/Os required to scan an entire table.

Before setting this parameter, you can use the following query on `V$SYSSTAT` to determine how often your application is performing full table scans:

```
SQL> SELECT name, value
2  FROM v$sysstat
3  WHERE name = 'table scans (long tables)';
```

```
NAME                              VALUE
---------------------------- ----------
table scans (long tables)           948
```

High values for this statistic indicate that your application is performing frequent full table scans and may benefit from an increase in the `DB_FILE_MULTIBLOCK_READ_COUNT` parameter.

The behavior of the Cost-based Optimizer is also affected by this parameter. If fewer I/Os are required to access all the rows of a table via a full table scan, then the CBO may now choose to access that table using a full table scan instead of a previously used index.
When using RAID striping, the value of `DB_FILE_MULTIBLOCK_READ_COUNT` should be set to a multiple of the stripe size.

Another view useful for monitoring full table scan operations is the `V$SESSION_LONGOPS` view. This view shows activity associated with selected long-running operations like snapshot refreshes, recovery operations, and parallel query jobs. Table 8.6 shows selected columns from the `V$SESSION_LONGOPS` view that can be used for monitoring full table scan activity.

**TABLE 8.6 Selected `V$SESSION_LONGOPS` Columns**

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SID</td>
<td>Session identifier of the user session running the long operation</td>
</tr>
<tr>
<td>SERIAL#</td>
<td>Serial number of the user session running the long operation</td>
</tr>
<tr>
<td>OPNAME</td>
<td>Name of the long operation being performed (TABLE SCAN in the case of full table scans)</td>
</tr>
<tr>
<td>START_TIME</td>
<td>Time that the long operation started</td>
</tr>
<tr>
<td>SO FAR</td>
<td>The total units of work completed so far</td>
</tr>
<tr>
<td>TOTAL WORK</td>
<td>The total units of work to be performed</td>
</tr>
</tbody>
</table>

Figure 8.7 shows a query on `V$SESSION_LONGOPS` that uses these columns.

**FIGURE 8.7 Query on `V$SESSION_LONGOPS`**

```sql
SQL> SELECT username, opname, ROUND(((totalwork-sofar)/totalwork)*100) "PCT_REMAINING" 
FROM v$session_longops
WHERE time_remaining > 0;
```

<table>
<thead>
<tr>
<th>USERNAME</th>
<th>OPNAME</th>
<th>PCT_REMAINING</th>
</tr>
</thead>
<tbody>
<tr>
<td>JIM</td>
<td>Sort Output</td>
<td>45.00</td>
</tr>
<tr>
<td>BRENTA</td>
<td>Table Scan</td>
<td>9.54</td>
</tr>
<tr>
<td>EMILY</td>
<td>SQL Execution</td>
<td>11.97</td>
</tr>
</tbody>
</table>
The output in Figure 8.7 shows that three long operations were underway when the query on `V$SESSION_LONGOPS` was issued. Occurrences of the second operation, `Table Scan`, are an indication that table scans are occurring in the application and can be tuned using the `init.ora` parameter `DB_FILE_MULTIBLOCK_READ_COUNT`.

The `Table Scan` operation will appear in `V$SESSION_LONGOPS` any time a process has scanned more than 10,000 Oracle blocks. Depending on the size of the table, this may, or may not, constitute a full table scan.

If frequent full table scans are occurring in the application, you can also use the `DBMS_APPLICATION_INFO.SET_SESSION_LONG_OPS` procedure to "register" a user session with the `V$SESSION_LONGOPS` view. Doing so will allow you to gain additional insights into the duration of long-running queries. In particular, this procedure can be used within a PL/SQL loop to allow you to monitor long-running application transactions and report back their progress to an application user.

See Chapter 3, "DBMS_APPLICATION_INFO," of Oracle9i Supplied PL/SQL Packages and Types Reference Release 1 (9.0.1) (Part Number A89852-02) for an example of how to use the `SET_SESSION_LOG_OPS` package in an anonymous PL/SQL block.

---

**Raw vs. Cooked: How Do You Like Your Disks?**

When a new disk is added to a Unix-based Oracle Server, the Unix system administrator has two choices for managing the I/O activities against disks:

- Create a Unix file system on the disk
- Leave the device in its raw format without a Unix file system on it

Disk partitions that contain file systems are referred to as "cooked" or `synchronous` devices. Disk partitions that do not contain file systems are referred to as "raw" or `asynchronous` devices. Which of these two device types to use on Oracle Servers has frequently been a source of hot debate among database administrators. The argument stems from the fact that,
Tuning DBW0 Performance

Database Writer (DBW0) is responsible for writing database buffers from the Database Buffer Cache to the database Datafiles. As you saw in Chapter 5, “Tuning the Database Buffer Cache,” DBW0 performs this write activity at database checkpoints and when user Server Processes are searching for free buffers in the Database Buffer Cache. Tuning Database Writer’s I/O activity is an important aspect of tuning the database’s overall disk I/O.

Measuring DBW0 I/O

The V$SYSTEM_EVENT and V$SYSSTAT dynamic performance views and the output from STATSPACK and REPORT.TXT can be used to determine whether DBW0 is experiencing any difficulty in meeting its obligations to fulfill write requests.

Using V$SYSTEM_EVENT to Measure Database Writer Performance

Occurrences of the buffer busy waits, db file parallel write, free buffer waits, and write complete waits system events in V$SYSTEM_EVENT
are both possible indicators of I/O performance problems related to DBW0. The event called buffer busy waits indicates that waits are being experienced for buffers in the Database Buffer Cache. Some of these waits may be due to inefficient writing of dirty buffers by DBW0. The event called db file parallel write indicates DBW0 may be experiencing waits when writing many blocks in parallel. These waits may be due to a slow device on which the Datafiles reside, or the fact that DBW0 cannot keep up with the write requests it is being asked to perform. Frequent free buffer waits events indicate that user Server Processes have been experiencing waits when moving dirty buffers to the Dirty List while searching the LRU for a free buffer. The write complete waits event indicates that user sessions have been experiencing waits for buffers to be written from the Database Buffer Cache by DBW0.

The following query shows how to determine what the values for these events are in your database:

```
SQL> SELECT event, total_waits, average_wait
2  FROM v$system_event
3  WHERE event IN ('buffer busy waits',
4          'db file parallel write',
5          'free buffer waits',
6          'write complete waits');
```

<table>
<thead>
<tr>
<th>EVENT</th>
<th>TOTAL_WAITS</th>
<th>AVERAGE_WAIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>db file parallel write</td>
<td>239</td>
<td>.034454825</td>
</tr>
</tbody>
</table>

This query shows that waits for DBW0 to perform parallel writes have occurred 239 times since instance startup. However, since the AVERAGE_WAIT column value is expressed in 100ths of a second, these waits are not yet significant in this particular case. Note that the lack of output for the buffer busy waits, free buffer waits, and write complete waits statistics indicates that this event has not occurred since instance startup.

High or steadily increasing values for buffer busy wait, db file parallel write, or write complete waits may indicate that DBW0 is not performing its write activities efficiently.
Using V$SYSSTAT to Measure Database Writer Performance

Occurrences of the statistics redo log space requests, DBWR buffers scanned, and DBWR lru scans in the V$SYSSTAT view are also useful for measuring the performance of DBW0. The statistic redo log space requests indicates that a wait occurred for a redo log to become available following a log switch. Since a database checkpoint occurs at a log switch, both DBW0 and LGWR have to complete their write activity from the Database Buffer Cache and Redo Log Buffer respectively before LGWR can start writing to the next redo log. If DBW0 is not writing the contents of the Database Buffer Cache fast enough, redo log space requests may result. The following query on V$SYSSTAT shows example output for this statistic:

```
SQL> SELECT name, value
       2  FROM v$sysstat
       3  WHERE name = 'redo log space requests';
```

```
NAME         VALUE
----------------------- ----------
redo log space requests          5
```

High or steadily increasing values for redo log space requests in V$SYSSTAT may indicate that DBW0 is not performing its write activities efficiently.

The DBWR buffers scanned and DBWR lru scans statistics in V$SYSSTAT can be also be used as indicators of Database Writer's effectiveness. The statistic DBWR buffers scanned indicates the total number of buffers in the Database Buffer Cache that were examined in order to find dirty buffers to write. Since DBW0 performs this action, DBW0 may perform writes ineffectively if it is busy examining the LRU List for dirty buffers when it is signaled to write. Dividing DBWR buffers scanned by the value for DBWR lru scans will yield the average number of buffers examined with each LRU scan. The following query on V$SYSSTAT shows this calculation:

```
SQL> SELECT scanned.value/scans.value
       2  'Avg. Num. Buffers Scanned'
       3  FROM v$sysstat scanned, v$sysstat scans
       4  WHERE scanned.name = 'DBWR buffers scanned'
       5  AND scans.name = 'DBWR lru scans';
```
Using this query, high or steadily increasing values for the Average Number of Buffers Scanned may indicate that DBW0 is not performing its write activities efficiently.

### Using STATSPACK to Measure Database Writer Performance

Much of the Database Writer performance information found in $V$SYSTEM\_EVENT and $V$SYSSTAT is also contained in the output of the STATSPACK feature. Figure 8.8 shows sample STATSPACK output that contains wait event information related to Database Writer.

The output in Figure 8.8 shows that 56 occurrences of the buffer busy waits event happened between the times when the two STATSPACK snapshots were taken. As discussed in the previous section, significant waits for the buffer busy waits, db file parallel write, free buffer waits, or write complete waits system events indicate that Database Writer may be causing a performance bottleneck. Figure 8.9 displays sample STATSPACK output that shows system statistics related to Database Writer.

**FIGURE 8.8** STATSPACK output showing Database Writer events

<table>
<thead>
<tr>
<th>Event</th>
<th>Waits</th>
<th>Time (s)</th>
<th>% Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>control file sequential read</td>
<td>79</td>
<td>0</td>
<td>28.2%</td>
</tr>
<tr>
<td>sync disk IO</td>
<td>139</td>
<td>0</td>
<td>25.27%</td>
</tr>
<tr>
<td>control file parallel write</td>
<td>214</td>
<td>0</td>
<td>25.08%</td>
</tr>
<tr>
<td>buffer busy wait</td>
<td>56</td>
<td>0</td>
<td>20.84%</td>
</tr>
<tr>
<td>log file parallel write</td>
<td>9</td>
<td>0</td>
<td>3.01%</td>
</tr>
</tbody>
</table>

The output in Figure 8.9 indicates that both DBWR lru scans and redo log space requests have occurred between the times when the two STATSPACK snapshots were taken. As discussed in the previous section, non-zero values for the redo log space requests, DBWR buffers scanned, and DBWR lru scans statistics are an indicator that Database Writer may benefit from tuning.
Using \textit{REPORT.TXT} to Measure Database Writer Performance

All of the events and statistics mentioned so far in this section can also be found in the output file \textit{REPORT.TXT}. Figures 8.10 and 8.11 show the sections of \textit{REPORT.TXT} that contain the information regarding the events from the \texttt{V$SYSTEM\_EVENT} view and statistics from the \texttt{V$SYSSTAT} view.
FIGURE 8.11 Sample REPORT.TXT showing V$SYSSTAT statistics

```
SQL> select n.name "Statistic",
        n.value/ (n.end - n.start) "Per Second",
        round(n.value/ (n.end - n.start) * 1000) "Per Second",
        n.value/ (n.end - n.start) "Per Logon",
        round(n.value/ (n.end - n.start) * 1000) "Per Logon"
        from v$sysstat n
        where n.name in ('DBWR checkpoint buffers used',
                         'DBWR checkpoints',
                         'DBWR checkpoint data written',
                         'DBWR buffer scanned',
                         'SQL*Net roundtrips to/from',
                         'Background checkpoints comp',
                         'Background checkpoints data',
                         'redo log space requests')
        order by n.name;
```

The presence of the buffer busy wait event and non-zero values for the DBWR buffer scanned and redo log space requests statistics in these figures also indicates that tuning may improve Database Writer’s performance.

Improving DBW0 I/O

Two init.ora parameters can be used to tune the performance of DBW0’s activities:

- DBWR_IO_SLAVES
- DB_WRITER PROCESSES

Each of these parameters, and techniques for using them to improve Database Writer I/O, are described in the following sections.

**Oracle Objective**

Configure the instance to use I/O slaves

**DBWR_IO_SLAVES**

This init.ora parameter specifies the number of Database Writer slave processes to start at instance startup. The default value for this parameter is 0.
The maximum value for this parameter is OS-dependent. The naming convention for the DBW0 slave background processes on a Unix server for an instance called PROD are shown here:

- First DBW0 I/O slave: ora_i101_PROD
- Second DBW0 I/O slave: ora_i102_PROD
- $n$th DBW0 I/O slave: ora_i$n$_PROD

Each component of the I/O slave’s process name is meaningful. The middle portion the process name, i103 for a process with the name ora_i103_PROD, can be interpreted this way: the first character, $i$, indicates that the process is a slave process, the second character, 1, indicates that this process using adapter (a pool of memory) number one, and the final two characters, 03, indicate that this is the third slave using this adapter.

Database Writer slave processes are similar to the actual DBW0 process itself, except they can only perform write operations, not move buffers from the LRU List to the Dirty List in the Database Buffer Cache as DBW0 does. The purpose of these slaves is to simulate asynchronous I/O on systems that only support synchronous I/O. See the next sidebar for an explanation of Asynchronous I/O.

After configuring the I/O slaves, DBW0 will coordinate assigning I/O requests to each of the slave processes. If all available slave processes are busy when an I/O request is made, then DBW0 will spawn additional processes—up to the limit specified by the value specified by DBWR_IO_SLAVES in init.ora. If no I/O slaves are available when a request is made, a wait will result.

Any time Database Writer spawns I/O slave processes, other background processes that perform disk I/O (i.e., Log Writer, Archiver, and the Recovery Manager utility) will also have I/O slaves created to support their I/O activities. For example, setting DBWR_IO_SLAVES to a non-zero value will spawn four Log Writer slaves as well.

Slaves for the Recovery Manager utility’s tape I/O activities can also be spawned by setting the BACKUP_TAPE_IO_SLAVES parameter in init.ora to a non-zero value.
How I/O Slaves Simulate Asynchronous I/O

As mentioned in the previous sidebar, devices can be configured as either raw or cooked. From the perspective of the DBW0, each read or write request to a cooked device is punctuated by a wait for the OS to signal that the operation is complete. Since each write request is blocked from continuing because of the wait for the OS’s signal, this type of I/O is also referred to as “blocking I/O.” A request to write four database blocks to a Datafile on a cooked device might look like the pattern shown in the following graphic.

Unlike the blocking I/O shown here, Database Writer does not have to wait for the write request to be processed before the next write request is serviced when using raw devices. Therefore, raw devices are generally faster at performing I/O operations than cooked devices because there is no wait for the OS to signal the completion of the I/O on asynchronous devices. Multiple DBW0 slaves allow the DBA to simulate asynchronous (raw) I/O on synchronous (cooked) I/O devices by interleaving read/write requests with read/write processing. The following graphic illustrates this process.

If your server O/S natively supports asynchronous I/O then you should probably utilize it. However, if your server’s asynchronous I/O capabilities prove inefficient for Oracle I/Os, you may want to tell Oracle to not use it and instead use I/O slaves. Oracle provides an init.ora parameter, DISK_ASYNCH_IO, just for this purpose. Setting this parameter to TRUE causes Oracle to use the O/S’s asynchronous I/O functionality. Setting the parameter to FALSE disables Oracle’s use of the O/S’s asynchronous I/O capabilities.
Chapter 8 • Tuning Disk I/O

**Oracle Objective**

Configure and use multiple DBW processors

**DB_WRITER PROCESSES**

While DBW0 slaves can help improve I/O performance, they are not able to perform all the functions of DBW0 (like managing the Dirty and LRU Lists in the Database Buffer Cache). If DBW0 needs assistance performing these tasks as well, you can start additional, full-fledged, Database Writer processes using the `DB_WRITER_PROCESSES` parameter in the `init.ora`. The default value for this parameter is 1. The maximum value for this parameter is 10. Once started, each Database Writer background process will be numbered sequentially: DBW0, DBW1, DBW2, and so on.

Multiple Database Writer processes generally have better throughput than Database Writer I/O slaves. For example, having five Database Writer processes with no I/O slaves will offer better performance than having one Database Writer process and four Database Writer I/O slaves. Which of these to use is dependent upon the types of waits indicated by the performance measurement queries shown in Chapter 5. If the Buffer Cache wait events are strictly related to DBW0 I/O activity, then starting I/O slaves is appropriate. If the Buffer cache wait events are occurring for activities related to internal Buffer Cache management (i.e., free buffer waits events), then starting multiple Database Writer processes to handle those structures would be more suitable.

---

**NOTE**

DBWR I/O slaves and multiple DBWR processes cannot be used simultaneously. If you specify a non-zero value for `DBWR_IO_SLAVES`, then the value for `DB_WRITER_PROCESSES` has no effect.

---

**Tuning Segment I/O**

As stated previously, Oracle segments (e.g., tables and indexes) store their data in tablespaces, which are made up of one or more of physical Datafiles. Each Datafile is in turn made up of individual database blocks. These blocks store the actual data for each segment and represent the smallest unit of I/O that can be performed against a Datafile. It is these blocks that are
copied into the Database Buffer Cache by each user’s Server Process, and are written by the DBW0 Oracle background process. When a segment is created, it is allocated a chunk of contiguous blocks, called an *extent*, from within a Datafile that is associated with the tablespace that the segment is stored in. The following list summarizes the Oracle storage hierarchy:

- Databases comprise one or more tablespaces
- Tablespaces use Datafiles to store segments
- Segments are made up of one or more extents
- Extents are made up of contiguous Oracle blocks
- Oracle blocks are made up of contiguous operating system blocks

This section of this chapter will discuss tuning considerations related to Oracle block and extent I/O.

**Understanding Oracle Blocks and Extents**

In Oracle9i, there are two Oracle block sizes to consider. The first is referred to as the *Primary Block Size*. The primary block size is set at database creation and is specified in bytes by the `init.ora` parameter `DB_BLOCK_SIZE`. The default value for `DB_BLOCK_SIZE` varies by platform. On most Unix systems, it is between 2K and 8K. Windows 2000 systems have a default block size of 2K. No matter the default, most systems can use 2K, 4K, 8K, 16K, 32K, and 64K block sizes. At a minimum, the blocks in the `SYSTEM` and `TEMP` tablespaces will use the primary block size. The only way to change the primary block size is to re-create the database.

The second block size to consider is the local block size associated with an individual tablespace. When a tablespace is created, a block size that is larger or smaller than the primary block size can be specified using the `BLOCKSIZE` keyword.

The value specified for `BLOCKSIZE` must match the value specified in the associated `DB_nK_CACHE_SIZE` parameter specified in the `init.ora`.

Any segment stored in that tablespace will use blocks equal to the local block size to store its data. If the `BLOCKSIZE` option is omitted at tablespace creation the tablespace will use the primary block size. You can use the `BLOCK_SIZE` column in `DBA_TABLESPACES` to determine the block size associated with each tablespace.
For optimal performance, the primary and local block sizes should be a multiple of the operating system block size, but not larger than the operating system’s maximum I/O size.

**Oracle Objectives**

- Describe the correct usage of extents and Oracle blocks
- Describe the use of Oracle block parameters

An extent is a collection of contiguous Oracle blocks. When a segment is created, it will be assigned at least one extent called the initial extent. When a segment is dropped, its extents are released back to the tablespace for use by another segment. Figure 8.12 shows the relationship between the following table, called EMPLOYEE, its initial extent of 250KB, the tablespace called APPL_DATA, and its associated Datafiles, appl_data01.dbf and appl_data02.dbf:

```sql
SQL> CREATE TABLE employee
2  ( emp_id   number,
3    last_name   varchar2(20),
4    first_name  varchar2(20),
5    start_date  date)
6  PCTFREE 20 PCTUSED 30
7  INITTRANS 5
8  STORAGE (initial 250K
9       next 250K
10      pctincrease 0
11      freelists 1)
12  TABLESPACE appl_data;

Table created.
```

The first block of this initial extent, called the header block, contains a road map to the locations of all the other blocks in the extent. If you were to zoom in on one block from the 32 blocks allocated to the initial 250KB extent of the EMPLOYEE table, you will see that it has the structure shown in Figure 8.13.
**FIGURE 8.12** Relationship between the EMPLOYEE table, initial extent, tablespace, and Datafiles

Initial Extent of EMPLOYEE Table  
(32 blocks × 8KB each block = 250KB)

<table>
<thead>
<tr>
<th>Header</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**FIGURE 8.13** Structure of an individual EMPLOYEE table block

Initial Extent of EMPLOYEE Table  
(32 blocks × 8KB each block = 250KB)

<table>
<thead>
<tr>
<th>Header</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Datafile  
/u02/oradata/PROD/appl_data01.dbf  
Datafile  
/u02/oradata/PROD/appl_data02.dbf
As shown in Figure 8.13, each block is divided into three sections:

- Block header area
- Reserved space
- Free space

These areas, used in conjunction with the Percent Free (PCTFREE) and Percent Used (PCTUSED) parameters, help determine how the block is used to store data.

**Block Header Space**

Every segment block uses some of its allocated space to store header information about the contents of that block. This header information includes the transaction slots specified by the INITRANS parameter at table creation, a directory of the rows that are contained within the block, and other general header information needed to manage the contents of the block. This block header information generally consumes between 50 to 200 bytes of the block’s total size.

**Reserved Space**

The parameter PCTFREE tells Oracle how much space to set aside in each block to store row updates. This value is specified as a percentage of the overall size of the block. For example, if you create a table with a PCTFREE setting of 20, then 20 percent of the block’s storage will be set aside to hold updated row information. Once a block is filled with data to the level of PCTFREE, the block will no longer accept new inserts, leaving the remaining space to store updates to the existing rows already stored in the block.

The process of removing a block from the list of available blocks is performed by the table’s Free List. You can think of a Free List as a clipboard that contains a listing of the block IDs that belong to a particular segment. Whenever a user’s Server Process wants to insert a new row into a segment, the Server Process consults the Free List to find the block ID of a block that is available to accept the insert. If the subsequent insert should cause the block to fill to above the cutoff determined by PCTFREE, then the block is taken off the Free List, and no other inserts will be made to that block.

A block stays off of the Free List until enough data is deleted so that the block’s available space falls below that specified by PCTUSED. When this occurs, the block will go back to the top of the Free List where it can be used again to store new inserts. The default values for PCTFREE and
PCTUSED are 10 percent and 40 percent respectively. PCTFREE and PCTUSED can be set at segment creation or changed after creation using the ALTER command. However, the ALTER command does not change the PCTFREE and PCTUSED for blocks that currently contain data. Only new blocks or blocks that are returned to the Free List will have the new setting for PCTFREE and PCTUSED.

Segments from which you expect little or no update can have their PCTFREE value set to a very low (even zero) value. This has the benefit of packing the blocks as fully as possible, causing fewer blocks to be read when accessing a segment’s data. There is no PCTUSED value associated with index blocks. Index blocks go back on the Free List only when they are completely empty.

**Free Space**

The remaining space in the segment block, after allowances for the block header and the PCTFREE reserved space, is free space that is used to store the row data for the segment. The number of rows that will fit into each block is dependent upon the size of the block and the average row length of the data stored in that segment.

Whenever noncontiguous free space existing in a block is large enough, in total, to hold an inserted or updated row, Oracle will coalesce the free space within the block prior to performing the row insert or update.

The default values for PCTFREE and PCTUSED are 10 and 40 percent respectively. You may adjust these values in certain situations. For example, PCTFREE can be set to zero if there will be no update activity on a table. Likewise PCTUSED can also be set to a higher value if there will be no delete activity on a table. Oracle recommends using these two formulas for determining the appropriate PCTFREE and PCTUSED values for a table:

\[
\text{PCTFREE} = 100 \times \frac{\text{Avg. Update Size (bytes)}}{\text{Avg. Row Length}}
\]

\[
\text{PCTUSED} = 100 - \text{PCTFREE} - 100 \times \frac{\text{Num. Rows in Table} \times \text{(Avg. Row Length)}}{\text{Block Size}}
\]
Improving Segment I/O

The overriding goal when tuning Segment I/O is to minimize the number of blocks that must be accessed in order to retrieve requested application data. As mentioned in Chapter 3, this can be largely achieved through the appropriate use of indexes and properly tuned SQL code. However, other key areas related to improving segment I/O include:

- The cost associated with dynamic extent allocation
- The performance impact of extent sizing
- The performance impact of block sizing
- How row chaining and migration affect performance
- The role of the High Water Mark during full table scans

Each of these areas will be addressed in the following sections.

DynamicExtentAllocationandPerformance

When all the available extents assigned to a segment are full, the next insert operation will cause the segment to acquire a new extent. On rapidly growing tables stored in traditional dictionary-managed tablespaces, this dynamic allocation of segment extents incurs undesirable I/O overhead. This I/O stems from the queries that Oracle must perform against the data dictionary in the SYSTEM tablespace and the actual acquisition of the extent space in the appropriate Datafile.

One way to avoid this issue completely is to store all application tables and indexes in locally managed tablespaces. Because locally managed tablespaces use bitmaps in the Datafile file headers to allocate free space instead of the data dictionary, the dynamic allocation of extents in locally managed tablespaces incurs very little recursive I/O. Instead, Oracle simply changes the bitmap values to indicate the status of the blocks within the Datafile.

Another way to avoid dynamic extent allocation is to identify tables and indexes that are close to needing an additional extent, and then proactively assigning them additional extents manually. The following query shows how you can identify all the segments that are already using 95 percent of the Oracle blocks allocated to them:

SQL> SELECT owner, table_name,
2       1-(empty_blocks/(empty_blocks+blocks))
3       '% Blocks Used'
4  FROM dba_tables
5  WHERE owner != 'SYS'
6  AND 1 - (empty_blocks/(empty_blocks+blocks)) > .95
7  ORDER BY 1;

<table>
<thead>
<tr>
<th>OWNER</th>
<th>TABLE_NAME</th>
<th>% Blocks Used</th>
</tr>
</thead>
<tbody>
<tr>
<td>APPL</td>
<td>SALES</td>
<td>1</td>
</tr>
<tr>
<td>APPL</td>
<td>EMPLOYEE</td>
<td>.96654275</td>
</tr>
<tr>
<td>APPL</td>
<td>DEPARTMENT</td>
<td>.96899225</td>
</tr>
</tbody>
</table>

This query indicates that the SALES table is currently using 100 percent of its allocated blocks. This means the SALES table will grow by one extent the next time a row is inserted into the table. If you expect more rows to be inserted into this table, you can use the following command to pre-allocate an additional extent to the table:

```
SQL> ALTER TABLE appl.sales
2  ALLOCATE EXTENT;
```

Adding this extent now will avoid the dynamic allocation that would have occurred during the next row insert.

For optimal performance, Oracle states that the maximum number of extents for a segment in a dictionary-managed tablespace should not exceed 1,000. Segments stored in locally managed tablespaces can have thousands of extents with no negative impact on performance.

**Extent Sizing and Performance**

Closely related to the dynamic allocation of extents is the size of each extent. Larger extent sizes offer slightly better performance than smaller extents because they are less likely to dynamically extend and can also have all their locations identified from a single block (called the *extent map*) stored in the header of the segment's first extent. Large extents also perform better during full table scan operations because fewer I/Os are required to read the extents if the *init.ora* parameter `DB_FILE_MULTIBLOCK_READ_COUNT` parameter is set properly. Occasionally a large table may also be so large that it would have hundreds of thousands of extents if the extents were small.

The disadvantages of large extents sizes are the potential for wasted space within tablespaces and the possibility that there may not be a large enough set of contiguous Oracle blocks available when an extent is required.
Block Size and Performance

Since extents are made up of contiguous blocks, database block size is closely related to I/O performance. The appropriate block size for your system will depend on your application, OS specifications, and available hardware. Generally, OLTP systems use smaller block sizes for these reasons:

- Small blocks provide better performance for the random-access nature of the OLTP systems.
- Small blocks reduce block contention, since each block contains fewer rows.
- Small blocks are better for storing the small rows that are common in OLTP systems.

However, small block sizes add to Database Buffer Cache overhead because more blocks must generally be accessed, since each block stores fewer rows.

Conversely, DSS systems use larger block sizes for these reasons:

- Large blocks pack more data and index entries into each block.
- Large blocks favor the sequential I/O common in most DSS systems.

However, larger block sizes also increase the likelihood of block contention and require larger Database Buffer Cache sizes to accommodate all the buffers required to achieve acceptable Database Buffer Cache hit ratios.

Block size also impacts two important block-related tuning issues: row chaining and row migration.

Oracle Objective

Describe and detect Chaining and Migration of Oracle blocks

Row Chaining

When a row that is inserted into a table exceeds the size of the database block, the row will spill over into two or more blocks. Whenever a row spans
multiple blocks in this manner, it is referred to as a \textit{chained row}. Row chaining is bad for performance because multiple blocks must be read to return a single row. The only way to fix a chained row is to either decrease the size of the insert or increase the Oracle block size.

\textbf{Row Migration}

\textit{Row migration} occurs when a previously inserted row is updated. If the update to the row causes the row to grow larger than the space available in the block specified by \texttt{PCTFREE}, Oracle moves (or migrates) the row to a new block. When migration occurs, a pointer is left at the original location, which points to the row’s new location in the new block. Row migration is bad for performance because Oracle must perform at least two I/Os (one on the original block, and one on the block referenced by the row pointer) in order to return a single row. Row migration can be minimized by setting \texttt{PCTFREE} to an appropriate value so updated rows find sufficient space in the same block to store the new data. There are two techniques for determining whether row chaining and/or migration are occurring in your database: examining the \texttt{CHAIN_CNT} column in \texttt{DBA_TABLES} and the presence of the \texttt{table fetch continued row} statistic in \texttt{V$SYSSTAT}.

\textbf{USING \texttt{DBA_TABLES} TO IDENTIFY ROW CHAINING AND MIGRATION}

The \texttt{ANALYZE} command populates the \texttt{CHAIN_CNT} column of \texttt{DBA_TABLES}, which is otherwise null. This column will indicate how many of a table’s rows are using more than one block to store their data. However, only the total number of rows using more than one block to store data is recorded. No distinction is made between the number of chained rows and the number of migrated rows.

\textbf{USING STATISTICS TO IDENTIFY ROW CHAINING AND MIGRATION}

A second method for identifying the existence of row chaining and migration in the database is by querying \texttt{V$SYSSTAT} for occurrences of the statistic \texttt{table fetch continued row}:

```
SQL> SELECT name, value
2  FROM V$SYSSTAT
3  WHERE name = 'table fetch continued row';
```

<table>
<thead>
<tr>
<th>NAME</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>table fetch continued row</td>
<td>450</td>
</tr>
</tbody>
</table>
This statistic is also found in the STATSPACK and REPORT.TXT output as shown in Figures 8.14 and 8.15.

**FIGURE 8.14** Sample STATSPACK report indicating Chaining/Migration

<table>
<thead>
<tr>
<th>Statistic</th>
<th>Total</th>
<th>0.0</th>
<th>0.7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Switch current to ker buffer</td>
<td>2</td>
<td>0.0</td>
<td>0.7</td>
</tr>
<tr>
<td>Table fetch by rowid</td>
<td>342</td>
<td>0.5</td>
<td>114.0</td>
</tr>
<tr>
<td>Table fetch continued row</td>
<td>147</td>
<td>0.2</td>
<td>8.3</td>
</tr>
<tr>
<td>Table scan blocked gotten</td>
<td>635</td>
<td>1.0</td>
<td>24.7</td>
</tr>
<tr>
<td>Table scan row gotten</td>
<td>2.422</td>
<td>11.3</td>
<td>2.427.7</td>
</tr>
</tbody>
</table>

**FIGURE 8.15** Sample REPORT.TXT indicating Chaining/Migration

<table>
<thead>
<tr>
<th>Statistic</th>
<th>Total</th>
<th>Per Transact</th>
<th>Per Logon</th>
<th>Per Second</th>
</tr>
</thead>
<tbody>
<tr>
<td>Switch current to ker buffer</td>
<td>1107</td>
<td>10.09</td>
<td>108.34</td>
<td>5.71</td>
</tr>
<tr>
<td>Table fetch by rowid</td>
<td>42245</td>
<td>4204.42</td>
<td>2402.63</td>
<td>2407.82</td>
</tr>
<tr>
<td>Table fetch continued row</td>
<td>10719</td>
<td>99.25</td>
<td>3339.68</td>
<td>51.59</td>
</tr>
<tr>
<td>Table scan blocked gotten</td>
<td>60981</td>
<td>592.05</td>
<td>7992.63</td>
<td>307.61</td>
</tr>
<tr>
<td>Table scan row gotten</td>
<td>3106708</td>
<td>10062.94</td>
<td>31809.75</td>
<td>8224.99</td>
</tr>
</tbody>
</table>

Since row migration only occurs during an update, it can be corrected by merely deleting and then reinserting the migrated rows. This can be accomplished by:

- Export, drop or truncate, and then re-import the table
- Use the `ALTER TABLE ... MOVE` command to rebuild the table
- Identify and then reinsert the migrated rows.

The following shows how to use this last technique to identify and remedy the migrated rows in a table called `SALES`.

The `ANALYZE TABLE` command is used to identify whether chained or migrated rows exist in the `SALES` table:

```sql
SQL> ANALYZE TABLE sales COMPUTE STATISTICS;
Table analyzed.

SQL> SELECT table_name, chain_cnt
2  FROM dba_tables
3  WHERE table_name = 'SALES';
```
Tuning Segment I/O

<table>
<thead>
<tr>
<th>TABLE_NAME</th>
<th>CHAIN_CNT</th>
</tr>
</thead>
<tbody>
<tr>
<td>SALES</td>
<td>5</td>
</tr>
</tbody>
</table>

The resulting \texttt{CHAIN_CNT} of 5 indicates that the \texttt{SALES} table has five rows that are chained and/or migrated.

This view does not distinguish between rows that are migrated and rows that are chained. Because chained rows can only be corrected by increasing the table's block size, the following steps will only fix the migrated rows.

Next, the Oracle-supplied script \texttt{UTLCHAIN.SQL} is used to build a table called \texttt{CHAINED_ROWS}. This table is then used to identify the chained and/or migrated rows by using the \texttt{ANALYZE} command with the \texttt{LIST CHAINED ROWS} parameter. The \texttt{UTLCHAIN.SQL} script, shown below, is located in \texttt{$\text{ORACLE_HOME/rdbms/admin}$} on Unix systems and \texttt{%ORACLE_HOME%\rdbms\admin} on Windows 2000 systems.

```
SQL> @$ORACLE_HOME/rdbms/admin/utlchain.sql
Table created.
SQL> ANALYZE TABLE sales LIST CHAINED ROWS;
Table analyzed.
```

Now the \texttt{CHAINED_ROWS} table can be queried to identify which rows are migrated and/or chained in the \texttt{SALES} table by using the \texttt{HEAD_ROWID} column of the \texttt{CHAINED_ROWS} table and the \texttt{ROWID} pseudo column of the \texttt{SALES} table:

```
SQL> SELECT owner_name, table_name, head_rowid
2  FROM chained_rows
3  WHERE table_name = 'SALES';
```

<table>
<thead>
<tr>
<th>OWNER_NAME</th>
<th>TABLE_NAME</th>
<th>HEAD_ROWID</th>
</tr>
</thead>
<tbody>
<tr>
<td>APPL</td>
<td>SALES</td>
<td>AAAEHvAAAGAAABnUAAD</td>
</tr>
<tr>
<td>APPL</td>
<td>SALES</td>
<td>AAAEHvAAAGAAABnUAEE</td>
</tr>
<tr>
<td>APPL</td>
<td>SALES</td>
<td>AAAEHvAAAGAAABnUAAF</td>
</tr>
<tr>
<td>APPL</td>
<td>SALES</td>
<td>AAAEHvAAAGAAABnUAAG</td>
</tr>
<tr>
<td>APPL</td>
<td>SALES</td>
<td>AAAEHvAAAGAAABnUAHH</td>
</tr>
</tbody>
</table>
Next, you need to copy the chained rows to a temporary table so that you can subsequently delete them from the SALES table, and then reinsert them into the SALES table:

```sql
SQL> CREATE TABLE temp
    2     AS SELECT *
    3     FROM sales
    4     WHERE rowid IN (SELECT head_rowid
    5                 FROM chained_rows);
Table created.
```

```sql
SQL> DELETE FROM sales
    2     WHERE rowid IN (SELECT head_rowid
    3                 FROM chained_rows);
5 rows deleted.
```

```sql
SQL> INSERT INTO sales
    2     SELECT * FROM temp;
5 rows created.
```

Finally, you can reanalyze the SALES table with the COMPUTE STATISTICS option. Any non-zero value remaining in the CHAIN_CNT column of DBA_TABLES indicates that those rows are chained, not migrated, since deleting and reinserting those rows did not fix the row:

```sql
SQL> ANALYZE TABLE sales COMPUTE STATISTICS;
Table analyzed.
```

```sql
SQL> SELECT table_name, chain_cnt
    2     FROM dba_tables
    3     WHERE table_name = 'SALES';
```

<table>
<thead>
<tr>
<th>TABLE_NAME</th>
<th>CHAIN_CNT</th>
</tr>
</thead>
<tbody>
<tr>
<td>SALES</td>
<td>1</td>
</tr>
</tbody>
</table>
In this example, one row was actually chained and not migrated, so deleting and reinserting the rows did not fix it.

**High Water Mark and Performance**

Another important consideration when tuning database I/O related to individual database blocks is the concept of *High Water Mark*. As a segment uses the database blocks allocated to its extents, the Oracle Server keeps track of the highest block ID that has ever been used to store segment data. This block ID is called the High Water Mark (HWM). Figure 8.16 illustrates this mechanism.

**FIGURE 8.16** High Water Mark mechanism

![Diagram of High Water Mark mechanism]

<table>
<thead>
<tr>
<th>Initial Extent</th>
<th>Next Extent</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>HWM</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Initial Extent</th>
<th>Next Extent</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>HWM</td>
</tr>
</tbody>
</table>

= Empty Block

= Block Containing Data

**Oracle Objectives**

*Explain space usage and the High Water Mark*

*Determine the High Water Mark*

The HWM is significant because a user’s Server Process reads all the segment blocks up to the HWM when performing a full table scan against a
segment. Since the HWM does not move when rows are deleted from a segment, many empty blocks may end up being scanned during a full table scan even though they no longer contain data. Additionally, if the segment is static (no additional rows are expected to be added to the segment), the space above the HWM will be wasted. After using the `ANALYZE` command, the `EMPTY_BLOCKS` column in `DBA_TABLES` will show the number of blocks that a table has above its HWM. This unused space can be released back to the tablespace using the `ALTER TABLE tablename DEALLOCATE UNUSED` command. The unused space above the HWM can also be determined using the `DBMS_SPACE.UNUSED_SPACE` procedure.

Despite its name, the `EMPTY_BLOCKS` column in `DBA_TABLES` does not show how many blocks below the HWM are empty. Instead, it simply indicates how many blocks are above the HWM.

The `BLOCKS` column in `DBA_TABLES` shows the number of blocks below the HWM. However, these blocks may, or may not, actually contain data so it is not a very accurate indication of the position of the HWM in relation to the table's actual data. In order to estimate how many of the blocks below the HWM are empty, you must use a query similar to the one shown in Figure 8.17.

**FIGURE 8.17** Query to identify empty blocks below HWM

```sql
SELECT BLOCKS "ACTUAL_BLOCKS_USED_BELOW_HWM",
       ROUND((t.avg_row_len * t.num_rows)/s.block_size,0) "EST_BLOCKS_NEEDED_BELOW_HWM",
       ROUND((t.avg_row_len * t.num_rows)/s.block_size,0) - Blocks "WASTED_BLOCKS_BELOW_HWM"
FROM dba_tables t, dba_tablespaces s
WHERE t.tablespace_name = s.tablespace_name
AND t.table_name = 'EMPLOYEES'
AND t.owner='HR'
```

<table>
<thead>
<tr>
<th>ACTUAL_BLOCKS_USED_BELOW_HWM</th>
<th>EST_BLOCKS_NEEDED_BELOW_HWM</th>
<th>WASTED_BLOCKS_BELOW_HWM</th>
</tr>
</thead>
<tbody>
<tr>
<td>2134</td>
<td>1872</td>
<td>262</td>
</tr>
</tbody>
</table>

The output in Figure 8.17 shows that, of the 2,134 blocks below the HWM in `EMPLOYEE` table, approximately 1,872 blocks are actually required to store rows in those blocks. This could mean that up to 262 empty blocks exist below the HWM for the `EMPLOYEE` table. Finding and fixing the HWM on tables which have a HWM that is disproportionately above the blocks that actually contain data can improve the performance of full table scans on those tables.
There are two techniques for moving a table’s HWM to the appropriate level:

- Export, drop or truncate, and then re-import the table
- Use the `ALTER TABLE ... MOVE` command to rebuild the table

### Tuning Sort I/O

Sorting occurs whenever data must be placed in a specified order. When a sort occurs in an Oracle instance, it can take place in one of two locations: in memory or on disk. Sorts in memory are the least expensive in terms of performance. Sorts to disk are the most expensive because of the extra overhead of the disk I/Os. In general, the primary tuning goal with regard to sorting is to minimize sort activity, but when it must be done, perform it in memory whenever possible. The first key to tuning sorting is to understand what types of actions cause sort activity to occur.

**Oracle Objective**

| Recover space from sparsely populated segments |

| Describe SQL operations which cause sorts |

The types of SQL statements that can cause database sorts to occur include the following:

- `ORDER BY`
- `GROUP BY`
- `SELECT DISTINCT`
- `UNION`
- `INTERSECT`
- `MINUS`
Oracle allows each user process to use a specified amount of memory for each sort operation. If a sort operation is smaller than this size, the sort will be performed in memory. If the sort operation is larger than this size, the sort is broken up into chunks called sort runs. Each chunk is then sorted individually, with the intermediate sorted chunks being written to the user’s temporary tablespace. Once all the chunks have been sorted, the final, sorted result is created when the separate sorted chunks are merged together and returned to the user.

The amount of memory set aside for each user’s Server Process to perform these sort operations is impacted by the following init.ora parameters:

- **SORT_AREA_SIZE**
- **SORT_AREA_RETAINED_SIZE**
- **PGA_AGGREGATE_TARGET**
- **WORKAREA_SIZE_POLICY**

Each of these init.ora parameters and their effect on sorting behavior is addressed in the following sections.

### SORT_AREA_SIZE

The default value for SORT_AREA_SIZE is OS-dependent. The minimum size for this parameter is equivalent to six Oracle blocks. The maximum size is OS-dependent. SORT_AREA_SIZE specifies how much memory each user’s Server Process should set aside to perform in-memory sort operations.

The SORT_AREA_SIZE parameter can be set in init.ora, or by issuing the ALTER SYSTEM SET SORT_AREA_SIZE=n DEFERRED command, or by using the ALTER SESSION SET SORT_AREA_SIZE=n command.
When using the Shared Server option, each user’s sort area is taken from the UGA, not the server’s main memory.

**Oracle Objective**

**SORT_AREA_RETAINED_SIZE**

By default, the value for `SORT_AREA_RETAINED_SIZE` is equal to the size of `SORT_AREA_SIZE`. The minimum size for this parameter is the equivalent of two Oracle blocks. The maximum size is limited to the value of `SORT_AREA_SIZE`. Once a sort operation is complete, if the sort area still contains sorted rows that need to be returned to the user, the user’s Server Process reduces the memory set aside for the final fetch to the value specified by `SORT_AREA_RETAINED_SIZE`.

The `SORT_AREA_RETAINED_SIZE` parameter can be set in `init.ora`, by using the `ALTER SYSTEM SET SORT_AREA_RETAINED_SIZE=n DEFERRED` command, or by using the `ALTER SESSION SET SORT_AREA_RETAINED_SIZE=n` command.

Depending on the degree of parallelism specified, sorts performed using the Oracle Parallel Query option can require several times the amount of memory specified by `SORT_AREA_SIZE`.

**PGA_AGGREGATE_TARGET**

After `SORT_AREA_SIZE` has been used to specify how much memory will be assigned for sorting to each user on a per-session basis, the parameter `PGA_AGGREGATE_TARGET` can be used to establish an upper boundary on the maximum amount of memory that all user processes can consume while performing database activities—including sorting. The default value for this `init.ora` parameter is zero. Valid values for `PGA_AGGREGATE_TARGET` are 10MB to 4000GB.
**WORKAREA_SIZE_POLICY**

This parameter is used to determine whether the overall amount of memory assigned to all user processes is managed explicitly or implicitly. When set to the default value of MANUAL, the size of each user's sort area will be equivalent to the value of SORT_AREA_SIZE for all users. When set to a value of AUTO, Oracle will automatically manage the overall memory allocations so that they do not exceed the target specified by PGA_AGGREGATE_TARGET.

The PGA_AGGREGATE_TARGET and WORKAREA_SIZE_POLICY init.ora parameters are new in Oracle9i.

If you wish to specify a target value for PGA_AGGREGATE_TARGET then you must set WORKAREA_SIZE_POLICY to a value of AUTO.

**Measuring Sort I/O**

Sort activity can be monitored using the V$SYSSTAT and V$SORT_SEGMENT dynamic data dictionary views, using the output from STATSPACK and REPORT.TXT, and using the output from the OEM Performance Manager.

<table>
<thead>
<tr>
<th><strong>Oracle Objectives</strong></th>
<th>Determine the number of sorts performed in memory</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Differentiate between disk and memory sorts</td>
</tr>
</tbody>
</table>

**Using V$SYSSTAT to Measure Sort Activity**

The V$SYSSTAT dynamic performance sort view has two statistics, sorts (memory) and sorts (disk), that can be used to monitor sort user activity. Using these two statistics, the following sample query shows how you can calculate an in-memory sort ratio:

```sql
SQL> SELECT mem.value/(disk.value + mem.value) 
       2   'In-memory Sort Ratio'
```
In-memory Sort Ratio
---------------------
0.99233

The output from this query shows that 99.23 percent of the sorts that have occurred since instance startup were performed in memory, and not to disk.

Oracle’s recommended tuning goal with regard to sorting is to have at least 95 percent of all sorts happen in memory.

Using STATSPACK Output to Measure Sort Activity

The report produced by the STATSPACK utility shows similar sort statistics. These statistics can be found in several areas of the STATSPACK report. The sample output in Figure 8.18 shows sort statistics found in the Load Profile section of a STATSPACK report.

**FIGURE 8.18** STATSPACK output showing sort activity
The output in Figure 8.18 indicates that .15 sorts per second and 32.67 sorts per transaction occurred during the time between the two STATSPACK snapshots. The average rows per sort, per transaction, during that same period were 63.49 rows. These statistics are useful for getting an overview of how much sort activity is taking place in the database. They do not, however, indicate how many of these sorts were in memory and how many required a write to disk. This information can be found in the Instance Efficiency Percentages section of a STATSPACK, shown in Figure 8.19.

**FIGURE 8.19** STATSPACK output showing in-memory sort activity

<table>
<thead>
<tr>
<th>Instance Efficiency Percentages (Target 100%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Buffer Miss %: 100.00</td>
</tr>
<tr>
<td>Buffer Hit %: 100.00</td>
</tr>
<tr>
<td>Library Hit %: 96.11</td>
</tr>
<tr>
<td>Execute to Parse %: 42.95</td>
</tr>
<tr>
<td>Parse CIO to Parse Filled %:</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Shared Pool Statistics: Begin</th>
<th>End</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory Usage %: 37.43</td>
<td>38.35</td>
</tr>
<tr>
<td>% SQL with execution &gt; 1: 53.75</td>
<td>56.41</td>
</tr>
<tr>
<td>% Memory for SQL w/execute: 63.87</td>
<td>79.79</td>
</tr>
</tbody>
</table>

The output in Figure 8.19 indicates that 97.96 percent of the sorts that occurred between the two STATSPACK snapshots occurred in memory. While this statistic is important, it does not tell you how many in-memory sorts and to-disk sorts occurring during the monitoring period; instead, this information can be found in the Instance Activity Stats section of a STATSPACK report, shown in Figure 8.20.

**FIGURE 8.20** STATSPACK output showing number and type of sorts

<table>
<thead>
<tr>
<th>Instance Activity Stats for DB: PROD</th>
<th>Instance: prod</th>
<th>Snapshots: 1-2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total</td>
<td>per Second</td>
<td>per Trans</td>
</tr>
<tr>
<td>-----------------</td>
<td>-----------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>Session use memory key</td>
<td>189,494</td>
<td>237.7</td>
</tr>
<tr>
<td>shared hash latch upgrades - no o</td>
<td>1,386</td>
<td>3.0</td>
</tr>
<tr>
<td>sorts (disk)</td>
<td>2</td>
<td>0.6</td>
</tr>
<tr>
<td>sorts (memory)</td>
<td>96</td>
<td>0.2</td>
</tr>
<tr>
<td>sorts (rows)</td>
<td>5,222</td>
<td>9.5</td>
</tr>
<tr>
<td>switch current to new buffer</td>
<td>2</td>
<td>0.0</td>
</tr>
</tbody>
</table>

...
The output in Figure 8.20 shows that 6,222 rows were sorted during the STATSPACK monitoring period and 98 sorts occurred: 96 in memory and 2 to disk.

**Using REPORT.TXT to Measure Sort Activity**

Figure 8.21 shows the information from REPORT.TXT that indicates the number of sorts that were performed in memory and to disk.

**FIGURE 8.21 REPORT.TXT sort statistics**

```
SQL> select sli.name "Statistic",
  2   sum(sli.change/"Transaction") "Per Transaction",
  3   sum(sli.change/('start_users + end_users)/2) "Per Logon",
  4   sum(sli.change/to_number(to_char(end_time, 'S'))-to_number(to_char(start_time, 'S'))) "Per Second"
  5   to_number(to_char(end_time, 'S')) "sorts (Disk)",
  6   to_number(to_char(start_time, 'S')) "sorts (Memory)"
  7   from statistics sli
  8   where sli.name='user commit'
  9   order by sli.name;

Statistic                  Total Per Transaction Per Logon Per Second
--------------------------------------------------------------------------------------
...                        ...                      ...                  ...                  ...
session using memory size  20788224              10672.82        10728.65         7204.87
shared hash latch upgrades 20877                  25.02             240.38            0.31
sorts (disk)               1                      0.00             0.13              0
sorts (memory)             822                    7.61             102.75            3.37
sorts (rows)               53945                  459.48           6743.13           229.35
swtch current to new buffer 1187                  19.89             146.38            5.71
table fetch by rowid       183265                 400.62           5605.83           207.91
...                        ...                      ...                  ...                  ...
```

The output in Figure 8.21 shows that 53,945 sorts took place between the time that UTLBSTAT.SQL and UTLESTAT.SQL were executed. This resulted in 823 sorts: 822 in memory and 1 to disk.

**Using OEM Performance Manager to Measure Sort Activity**

The Performance Manager component of the Oracle Enterprise Manager Diagnostics Pack includes several graphical representations of sort performance. Figure 8.22 shows a sample Performance Manager screen that displays the instance's in-memory sort activity.

The output shown in Figure 8.22 indicates that majority of sort activity occurred in memory. However, two spikes occurred where in-memory sort activity dropped to approximately 90 percent.
Chapter 8 • Tuning Disk I/O

Figure 8.22 Performance Manager Sort Statistics

Improving Sort I/O

Sort activity can cause excessive I/O when performed to disk instead of memory, so you should try to tune sort activity so that its impact is minimized. There are several possible methods of improving sort performance, including:

- Avoiding SQL statements that cause sorts
- Adjusting the init.ora parameters related to sorting to encourage in-memory sorts
- Making proper use of temporary tablespaces
- Improving the reads related to sort I/O

Each of these techniques is discussed in the following sections.

Oracle Objective

Reduce total sorts and disk sorts

Avoid Sorts

One way to improve sort performance is to minimize the number of sorts being performed by application code, ad-hoc queries, and DDL activities.
A brief list of SQL statements that can cause sorts appeared earlier in this chapter. Some techniques for avoiding the sorts associated with these SQL statements are explained below.

**SQL Statement Syntax**

If the desired output is not negatively affected, use of the `UNION ALL` operator instead of the `UNION` operator will avoid the sort normally required to eliminate the overlapping values found in the two unioned SQL queries. Avoiding the use of the `INTERSECT`, `MINUS`, and `DISTINCT` keywords will yield similar results.

**Appropriate Use of Indexes**

Indexing columns that are referenced by `ORDER BY` and `GROUP BY` clauses in application SQL statements can minimize unnecessary sorting. Making sure that all foreign key columns are indexed will also help reduce sorting when tables are joined via primary and foreign key columns. This will help encourage the optimizer to perform nested loop joins instead of sort-merge joins when joining tables.

**Index Creation Overhead**

If a table being indexed already has its table data stored in index order, you can eliminate the sort that normally occurs during index creation by using the `NOSORT` option of the `CREATE INDEX` command.

**Statistics Calculation Overhead**

Sorts can occur whenever table and index statistics are gathered. Consider using the `ESTIMATE` option instead of `COMPUTE` when gathering table and index statistics to minimize the overhead associated with this process. Statistic gathering overhead can be further reduced by gathering statistics for only those columns that are relevant to the application SQL by using the `ANALYZE ... FOR COLUMNS` command.

### Make It Bigger

Another technique for improving I/O related to sorting is to minimize the number of sorts that are done to disk by increasing the value of `SORT_AREA_SIZE` in the `init.ora`. Care must be taken when increasing this parameter, however. While the amount of memory specified by `SORT_AREA_SIZE` is not allocated to a user’s session until they initiate a sort, the server’s available memory must be sufficient to accommodate the user’s sort area while the sort is being processed. In particular, if `SORT_AREA_SIZE` is set to a large value and many users are sorting simultaneously, the demands placed on the server’s memory may adversely impact performance until the memory allocated to those sorts is released when the sorts are...
complete. Decreasing the size of `SORT_AREA RETAINED_SIZE` when `SORT_AREA_SIZE` is increased will help minimize the problem of excessive memory usage related to sorting.

Special consideration must be given before increasing `SORT_AREA_SIZE` when the Parallel Query feature is being used. In Parallel Query environments a single SQL statement can require memory equal to `SORT_AREA_SIZE \times 2 \times` Degree of Parallelism specified.

Oracle states that the default value for `SORT_AREA_SIZE` is adequate for most OLTP systems. Oracle also advises that a `SORT_AREA_SIZE` of 1MB is generally optimal for systems that use the Oracle Parallel Query option.

**Oracle Objective**

Create and monitor temporary tablespaces

**Creating Temporary Tablespaces**

When a user’s Server Process writes a sort chunk to disk, it writes the data to the user’s temporary tablespace. This tablespace, although it is referred to as the user’s temporary tablespace, can have the tablespace attribute of being either permanent or temporary. Table 8.7 compares temporary and permanent tablespaces.

**Table 8.7** Comparison of Temporary and Permanent Tablespaces

<table>
<thead>
<tr>
<th>Tablespace Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Permanent</td>
<td>Can contain permanent segments like tables and indexes, as well as multiple temporary sort segments that are owned by each individual user’s Server Process</td>
</tr>
<tr>
<td>Temporary</td>
<td>Can contain only a single temporary segment that is shared by all users performing sorts to disk</td>
</tr>
</tbody>
</table>
You can use the following query on the DBA_TABLESPACES data dictionary view to determine whether a tablespace is permanent or temporary:

```
SQL> SELECT tablespace_name, contents
  2  FROM dba_tablespaces;
```

<table>
<thead>
<tr>
<th>TABLESPACE_NAME</th>
<th>CONTENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSTEM</td>
<td>PERMANENT</td>
</tr>
<tr>
<td>TEMP</td>
<td>TEMPORARY</td>
</tr>
<tr>
<td>APPL_DATA</td>
<td>PERMANENT</td>
</tr>
<tr>
<td>OEMGR</td>
<td>PERMANENT</td>
</tr>
<tr>
<td>RBS</td>
<td>PERMANENT</td>
</tr>
<tr>
<td>TOOLS</td>
<td>PERMANENT</td>
</tr>
<tr>
<td>USERS</td>
<td>PERMANENT</td>
</tr>
<tr>
<td>WEBDB</td>
<td>PERMANENT</td>
</tr>
<tr>
<td>APPL_IDX</td>
<td>PERMANENT</td>
</tr>
</tbody>
</table>

Further extending the idea of temporary tablespaces is the concept of tempfiles. Tempfiles are similar to Datafiles, except that they are used exclusively to store sort segment data written to temporary tablespaces. The following SQL command can be used to create a locally managed, temporary tablespace using a tempfile:

```
SQL> CREATE TEMPORARY TABLESPACE TEMP
   2  TEMPFILE '/u04/oradata/PROD/temp01.dbf' SIZE 500M
   3  AUTOEXTEND ON NEXT 50M MAXSIZE 2000M
   4  EXTENT MANAGEMENT LOCAL
   5  UNIFORM SIZE 64K;
```

The UNIFORM SIZE value specified for the temporary tablespace should be equal to, or a multiple of, SORT_AREA_SIZE + 1 block. This allows each sort chunk to be written with one I/O.

Striping the tempfile on multiple devices as described earlier in this chapter can provide excellent performance improvements. Once created, the database tempfiles can be monitored using the V$TEMPFILE and DBA_TEMP_FILES...
data dictionary views. The contents of these views is similar to the contents of the V$DATAFILE and DBA_DATA_FILES views.

Oracle9i also allows you to specify a temporary tablespace as the default temporary tablespace for all database users using the ALTER DATABASE command:

```
SQL> ALTER DATABASE DEFAULT TEMPORARY TABLESPACE temp;
```

The above command would make the TEMP (instead of SYSTEM) tablespace the default temporary tablespace for all database users.

---

**Note**

Designating a non-SYSTEM tablespace as the default temporary tablespace has some implications for database management. The default temporary tablespace must be of the type temporary (i.e. cannot be permanent) and cannot be converted to permanent once it has been designated as the default. Additionally, you cannot drop the default temporary tablespace until another has been created because doing so would leave the database with nowhere to perform to-disk sorts. Likewise, the default temporary tablespace also cannot be taken offline.

---

**Tip**

Because temporary tablespaces cannot contain any permanent objects, they do not need to be included in the database backup strategy.

If the database is not using temporary tablespaces for sorting, each user who performs a sort must build, and then drop, a sort segment within their temporary tablespace. This dynamic management of individual sort segments is expensive both in terms of I/O and recursive data dictionary calls.

Unlike a sort segment stored in a permanent tablespace, the sort segment in the temporary tablespace is not dropped when the user’s sort completes. Instead, the first sort operation following instance startup creates a sort segment that remains in the temporary tablespace for reuse by subsequent users who also perform sorts to disk. This sort segment will remain in the temporary tablespace until instance shutdown. You can monitor the size and usage of the sort segment using the V$SORT_SEGMENT and V$SORT_USAGE dynamic performance views.
By default, the sort segment created in the temporary tablespace has a MAXEXTENTS value of UNLIMITED.

**Using V$SORT_SEGMENT**

The V$SORT_SEGMENT view allows you to monitor the size and growth of the sort segment that resides in the temporary tablespace. The sort segment will grow dynamically as users place demands on it. The following query shows three of the more useful columns in this view:

```
SQL> SELECT tablespace_name, current_users, max_sort_blocks
       FROM v$sort_segment;
```

<table>
<thead>
<tr>
<th>TABLESPACE_NAME</th>
<th>CURRENT_USERS</th>
<th>MAX_SORT_BLOCKS</th>
</tr>
</thead>
<tbody>
<tr>
<td>TEMP</td>
<td>11</td>
<td>300</td>
</tr>
</tbody>
</table>

This query output shows that 11 users are currently using the sort segment stored in the TEMP tablespace. In addition, the largest single sort by an individual user was the equivalent of 300 Oracle blocks in size. This last statistic in particular, MAX_SORT_BLOCKS, can be useful when trying to determine what the optimal setting for SORT_AREA_SIZE and SORT_AREA_RETAINED_SIZE should be. By setting these init.ora parameters to at least MAX_SORT_BLOCKS * DB_BLOCK_SIZE, you can accommodate more sorts in memory and reduce I/O.

**USING V$SORT_USAGE**

While V$SORT_SEGMENT shows overall sort operations well, it does not allow you to see which individual users are causing large sorts to disk. However, the V$SORT_USAGE view does include a USER column which can be used to monitor sort activity by user as shown by the following query:

```
SQL> SELECT user, tablespace, blocks
       FROM v$sort_usage
       ORDER BY blocks;
```
<table>
<thead>
<tr>
<th>USER</th>
<th>TABLESPACE</th>
<th>BLOCKS</th>
</tr>
</thead>
<tbody>
<tr>
<td>APPS</td>
<td>TEMP</td>
<td>200</td>
</tr>
<tr>
<td>JOE</td>
<td>TEMP</td>
<td>110</td>
</tr>
<tr>
<td>BRENDA</td>
<td>TEMP</td>
<td>20</td>
</tr>
</tbody>
</table>

The TABLESPACE column tells you in which tablespace the sorts are being performed. The BLOCKS column indicates size of the sort activity in Oracle blocks. Monitoring this value can also be useful when setting the init.ora parameters SORT_AREA_SIZE and SORT_AREA_RETAINED_SIZE.

The V$SORT_USAGE view only contains entries when at least one sort is in progress. If you query this view while no users are connected, or when no sort activity is being performed, no rows will be returned.

### Sorting: The Whole Story

Knowing which user is performing the most sorting in your database is useful information. However, it is generally even more useful to know exactly what SQL operations they are performing to cause the sorts. With this information, you may be able to identify a SQL statement that could have its sorting eliminated by creating a new index or rewriting some application code.

The following query on V$SESSION, V$SQLAREA, and V$SORT_USAGE shows how the SQL being issued by users with high sort activity can be identified:

```
SQL> SELECT sess.username,
    2       sql.sql_text,
    3       sort.blocks
    4  FROM v$session sess,
    5       v$sqltext sql,
    6       v$sort_usage sort
    7  WHERE sess.serial# = sort.session_num
    8  AND   sort.sqladdr = sql.address
    9  AND   sort.sqlhash = sql.hash_value
   10  AND   sort.blocks > 200;
```
One of Oracle's basic tenets is never to allow a user to view the data from another user's uncommitted transaction. In order to meet this objective, Oracle must make use of rollback segments to store the before-image of changed data whenever an application user performs DML activity. Instead of the data contained in the uncommitted dirty buffers in the Buffer Cache, Oracle uses these before-images to display the original version of the modified data when other application users request it. In this manner, rollback segments play a critical role in every database DML transaction. Because of this, rollback segments should be examined for possible performance enhancement. The following section provides additional details on how rollback segments are used to process user transactions.

**Understanding Rollback Segments**

When a user starts a DML transaction, the before-image of the changed data is buffered in the Database Buffer Cache in the SGA. Copies of these buffers are also written to a rollback segment, which is stored in a tablespace. This tablespace is in turn made up of one or more physical Datafiles. The before-image data stored in the rollback segment is used for three important purposes:

- It can be used to restore the original state of the data if the user performing the DML command issues a ROLLBACK command.
- It provides a read-consistent view of the changed data to other users who access the same data prior to the DML user issuing a COMMIT command.
- It is used during instance recovery to undo uncommitted transactions that were in progress just prior to an instance failure.
Rollback segments are also referred to as **undo segments** because they are used to **undo** a transaction when a **ROLLBACK** command is issued.

Like any other database segment, rollback segments are made up of extents, which are in turn comprised of five or more contiguous Oracle blocks. Within each rollback segment, Oracle uses the extents in a circular fashion until the rollback segment is full. Once the rollback segment is full, no new transactions will be assigned to it until some of the rollback segment space is released when a user issues a **COMMIT** or **ROLLBACK** command. A single rollback segment can store the before-images for several different transactions. However, a transaction writes its before-image information to only one rollback segment. Figure 8.23 illustrates this concept for a rollback segment called RBS01, which is made up of eight extents.

**FIGURE 8.23** How transactions use rollback segments

Figure 8.23 shows that two transactions, A and B, are both storing their before-images in extent 1 of rollback segment RBS01, while a larger transaction (C) is using extents 2 through 6 to store its undo information. Extents 7 and 8 are currently inactive.
Once a transaction is assigned to a rollback segment, the transaction never switches to a different rollback segment, even if the original rollback segment was not the most appropriate choice. If the before-image of a transaction that starts in one extent of a rollback segment grows to fill that extent, the transaction will wrap into the adjacent extent and continue writing before-image information to that extent. Figure 8.24 illustrates this concept for transaction C described above.

**FIGURE 8.24** Transaction before-image wrapping to next extent

If the adjacent extent already contains before-image information for some other active transaction, Oracle will not overwrite that information, but will instead add a new extent to the rollback segment in order to store the transaction’s growing before-image. Figure 8.25 shows how this extent allocation occurs when a large transaction needs additional space to store its before-image.

Figure 8.25 shows how Transaction C, which needed additional space to store its before-image, was unable to wrap from extent 6 to extent 7 because Transaction D was already using extent 7 to store its before-image. Oracle also did not allow Transaction C to skip over extent 7 and use the next empty extent (8) instead. This caused a new extent (N) to be added, into which transaction C then wrapped to store its before-image.
The frequency with which transactions cause wrapping and extending of rollback segments will be key to the tuning of their performance.

**Figure 8.25** Dynamic rollback segment extent allocation

Unless you ask for a specific rollback segment using the `SET TRANSACTION USE ROLLBACK SEGMENT` command, the Oracle Server assigns transactions to rollback segments in such a way as to try and have each rollback segment manage about the same number of transactions. The total number of transactions a rollback segment can handle is dependent on the Oracle block size of your database.

Table columns that use Large Object (LOB) datatypes will not use rollback segments to store LOB before-images. These segments use space associated with the `PCTVERSION` value, assigned to the segment at creation to store LOB before-images.
Rollback segment tuning can be one of the most elusive aspects of database administration. Even when rollback activity has reached a steady state and no problems are apparent, one large transaction can cause a rollback segment error to occur. This can lead to frustrating rollback segment problems that never completely go away. The goals of rollback-segment tuning usually involve the following:

- Making sure that database users always find a rollback segment to store their transaction before-images without experiencing a wait
- Making sure that database users always get the read-consistent view they need to complete their transactions
- Making sure that the database rollback segments do not cause unnecessary I/O

Every database contains at least one rollback segment, which is the system rollback segment. Once other rollback segments are created in the database, this segment is used only for data dictionary read consistency and transaction control.

**Measuring Rollback Segment I/O**

In general, there are four areas that you should monitor when trying to tune the performance of rollback segments:

- Contention for the header of the rollback segments
- Contention for the rollback segment extents
- Wrapping of transactions from one extent to another within rollback segments
- Dynamic extent allocation by the rollback segments

Each of the above undo segment performance indicators can be monitored using `V$` dynamic performance views, output from STATSPACK and `REPORT.TXT`, and the OEM Performance Manager GUI tool.
Because rollback segments can be used to “undo” a user’s DML and restore the original data using the before-image, many of the statistics in these views use the term *undo* rather than *rollback segment* in their names.

---

**Oracle Objective**

Use the dynamic performance views to check rollback segment performance

---

**Measuring Rollback Segment Header Contention**

The Oracle Server uses a transaction table stored in the header block of each rollback segment to track the transactions that use the segment to store their before-images. This header block is generally cached in the Database Buffer Cache so that all users can access it when trying to store their transaction before-images. On a busy OLTP system, users may experience a wait for access to these rollback *segment header* blocks, thereby causing their transaction performance to decline.

You can measure the frequency with which these waits are occurring using the `V$SYSTEM_EVENT`, `V$WAITSTAT`, and `V$ROLLSTAT` dynamic performance views as well as the output from STATSPACK and REPORT.TXT.

**Using `V$SYSTEM_EVENT` to Measure Undo Header Contention**

The `V$SYSTEM_EVENT` view tracks performance related information on rollbacks segments via the `undo segment tx slot` statistic. The following query shows how to display this information:

```sql
SQL> SELECT event, total_waits, 
       2         time_waited, average_wait 
       3  FROM v$system_event 
       4  WHERE event like '%undo%';
```

<table>
<thead>
<tr>
<th>EVENT</th>
<th>TOTAL_WAITS</th>
<th>TIME_WAITED</th>
<th>AVERAGE_WAIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>undo segment tx slot</td>
<td>28070</td>
<td>18</td>
<td>.000641254</td>
</tr>
</tbody>
</table>
This query indicates that the average wait for access to the header blocks of the rollback segments was .0006 100ths seconds, essentially zero.

Ideally, the value of the AVERAGE_WAIT statistic should be consistently at or near zero.

**Using V$WAITSTAT to Measure Undo Header Contention**

The V$WAITSTAT view contains information on block contention statistics. Included in these statistics are counts for the number of times application users have experienced a wait for access to the header block of rollback segments. The following query shows how this information appears in the V$WAITSTAT view:

```sql
SQL> SELECT class, count
2  FROM v$waitstat
3  WHERE class IN ('undo header',
4    'system undo header')
```

<table>
<thead>
<tr>
<th>CLASS</th>
<th>COUNT</th>
</tr>
</thead>
<tbody>
<tr>
<td>system undo header</td>
<td>0</td>
</tr>
<tr>
<td>undo header</td>
<td>12</td>
</tr>
</tbody>
</table>

The statistic for **system undo header** relates to the system rollback segment. The **undo header** statistic is for the non-system rollback segments. This query indicates that there was no contention for the system rollback segment’s header, but that there were 12 occasions where waits occurred for access the headers of non-system rollback segments.

Ideally, the counts for these waits should be consistently at or near zero.

**Using V$ROLLSTAT to Measure Undo Header Contention**

The V$ROLLSTAT view contains detailed information regarding the behavior of the rollback segments in the database. In particular, the columns USN, GETS and WAITS are particularly useful for measuring contention for the
rollback segment’s header. Table 8.8 shows the descriptions of each of these columns.

**Table 8.8** \textit{V$ROLLSTAT} Columns Used to Measure Header Contention

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>USN</td>
<td>Undo Segment Number, the internal system-generated number assigned to the rollback segment</td>
</tr>
<tr>
<td>GETS</td>
<td>Number of times a user’s Server Process needed to access the rollback segment header and did so successfully</td>
</tr>
<tr>
<td>WAITS</td>
<td>Number of times a user Server Process needed to access the rollback segment header and experienced a wait</td>
</tr>
</tbody>
</table>

Using these columns, and joining \textit{V$ROLLSTAT} to the \textit{V$ROLLNAME} view on the common column USN, each rollback segment’s name can be obtained, allowing you to construct a query that will return the get ratio for each rollback segment header in your database:

```sql
SQL> SELECT n.name, s.usn,
       2        DECODE (s.waits,0,1
       3                     ,1-(s.waits/s.gets))
       4 'RBS Header Get Ratio'
       5 FROM v$rollstat s, v$rollname n
       6 WHERE s.usn = n.usn
       7 ORDER BY usn;
```

<table>
<thead>
<tr>
<th>NAME</th>
<th>USN</th>
<th>RBS Header Get Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSTEM</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>RBS01</td>
<td>1</td>
<td>.97</td>
</tr>
<tr>
<td>RBS02</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>RBS03</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>RBS04</td>
<td>4</td>
<td>.8425</td>
</tr>
<tr>
<td>RBS05</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>RBSBIG</td>
<td>6</td>
<td>1</td>
</tr>
</tbody>
</table>
The output from this query shows that most of the rollback segments are not experiencing any waits for access to their headers. Only RBS01 and RBS04 experienced waits at all.

Oracle recommends that a well-tuned OLTP system have a rollback segment header get ratio of 95 percent or higher for all rollback segments.

**Using STATSPACK Output to Measure Undo Header Contention**

The output from the STATSPACK utility also produces useful information for measuring the rollback segment header contention. Figure 8.26 shows sample output from a STATSPACK report that includes rollback segment performance statistics.

**FIGURE 8.26 STATSPACK output showing undo statistics**

![STATSPACK output showing undo statistics](image)

The *Pct Waits* column in the output in Figure 8.26 shows that no transactions experienced waits while trying to access the rollback segments headers.

**Using REPORT.TXT to Measure Undo Header Contention**

Rollback segment header contention statistics are also written to the REPORT.TXT file. Figures 8.27 and 8.28 show the sections from REPORT.TXT, which contain the information of rollback segment header contention.
FIGURE 8.27 REPORT.TXT showing undo wait statistics

The output in Figure 8.27 shows that waits for access to non-system rollback segment headers occurred twice between the time UTLBSTAT.SQL and UTLESTAT.SQL were executed.

Because the output in Figure 8.28 shows that the value for TRANS_TBL_WAITS waits were all zero during the monitoring period, this indicates that no contention occurred while trying to access the undo segment headers.

Rollback Segment Extent Contention

Even when there are no waits occurring for access to the rollback segment header blocks, there may still be waits occurring for access to the blocks
within the individual extents of each rollback segment. In addition to showing information on waits for rollback segment header blocks, the V$WAITSTAT view also contains statistics on waits for access to rollback segment extent blocks. The following query shows how to determine whether these waits are occurring in the database:

```
SQL> SELECT class, count
  2  FROM v$waitstat
  3  WHERE class IN ('undo block',
  4       'system undo block')
```

<table>
<thead>
<tr>
<th>CLASS</th>
<th>COUNT</th>
</tr>
</thead>
<tbody>
<tr>
<td>system undo block</td>
<td>0</td>
</tr>
<tr>
<td>undo block</td>
<td>22</td>
</tr>
</tbody>
</table>

The output from this query shows that your application users experienced 22 waits for access to rollback segment blocks since instance startup. You can determine the percentage of the total number of requests that waited by querying V$SYSSTAT for the consistent gets statistic and then calculating a ratio:

```
SQL> SELECT value
  2  FROM v$sysstat
  3  WHERE name = 'consistent gets';
```

```
VALUE
-------
7804633
```

The consistent gets statistic refers to the number of times rollback segment extent blocks were accessed in the Database Buffer Cache. By using the values from these two queries, you can calculate a wait ratio for access to rollback segment extent blocks: 22 waits with 7,804,633 attempts to access a rollback segment block = .000003 or .0003 percent wait ratio.

If the wait ratio for rollback segment extent blocks exceeds 1 percent, Oracle recommends that you consider tuning your rollback segments.
Rollback Segment Transaction Wrapping

When a transaction’s before-image exceeds the size of the extent to which it has been allocated, the transaction will wrap into the next extent if it is available. This dynamic wrapping incurs a small I/O cost and should be avoided if possible. You can examine the V$ROLLSTAT dynamic performance view and the output from STATSPACK and REPORT.TXT to determine whether transaction wrapping is occurring in your rollback segments.

Using V$ROLLSTAT to Measure Rollback Segment Wrapping

You can query the WRAPS column of V$ROLLSTAT to see how often transactions have wrapped from one extent to another since instance startup:

```
SQL> SELECT n.name, s.usn, s.wraps
2  FROM v$rollname n, v$rollstat s
3  WHERE n.usn = s.usn;
```

<table>
<thead>
<tr>
<th>NAME</th>
<th>USN</th>
<th>WRAPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSTEM</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>RBS01</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>RBS02</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>RBS03</td>
<td>3</td>
<td>22</td>
</tr>
<tr>
<td>RBS04</td>
<td>4</td>
<td>35</td>
</tr>
<tr>
<td>RBS05</td>
<td>5</td>
<td>8</td>
</tr>
<tr>
<td>RBSBIG</td>
<td>6</td>
<td>0</td>
</tr>
</tbody>
</table>

The output from this query shows that undo segments RBS01, RBS03, RBS04, and RBS05 all serviced transactions whose before-images did not fit into a single rollback segment extent. This caused those transactions to wrap to the rollback segment’s next extent 10, 22, 35, and 8 times, respectively.

Frequent wrapping indicates that the extent sizes of each rollback segment may be too small.

Using STATSPACK Output to Measure Rollback Segment Wrapping

The output from the STATSPACK utility also shows undo segment wrapping activity. Figure 8.29 shows a sample section from a STATSPACK report that includes rollback segment wrapping information.
FIGURE 8.29  STATSPACK output showing undo segment wrapping

This output indicates that some rollback segments, such as number 17, did experience problems with extent wrapping. Undo segment 17 had 21 transactions whose before image was too big to fit into the extent to which it had been assigned. This means that the transaction had to wrap (denoted by Wraps column) into the next extent of the rollback segment in order to find the space it needed to hold all the transaction’s undo information.

Using REPORT.TXT to Measure Rollback Segment Wrapping

Extent wrapping statistics are also available in the output from REPORT.TXT as shown in Figure 8.30.

FIGURE 8.30  REPORT.TXT Output Showing Undo Segment Wrapping

SQL> set lines 158;
SQL> set numwidth 15;
SQL> select "segment_number", "trans_tbl_get", "trans_tbl_waits", "undo_bytes_written", "segment_size_bytes", "shrink", "wraps" from stats_roll
SQL> where "rollback_segment_number" = 17;

<table>
<thead>
<tr>
<th>SEGMENT_NUMBER</th>
<th>TRANS_TBL_GETS</th>
<th>TRANS_TBL_WAITS</th>
<th>UNDO(Bytes)</th>
<th>SEGMENT_SIZE(Bytes)</th>
<th>SHRINK</th>
<th>WRAPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>102</td>
<td>0</td>
<td>104964</td>
<td>519400</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>694</td>
<td>0</td>
<td>604600</td>
<td>770240</td>
<td>0</td>
<td>11</td>
</tr>
<tr>
<td>3</td>
<td>363</td>
<td>0</td>
<td>102400</td>
<td>108416</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>375</td>
<td>0</td>
<td>670460</td>
<td>809112</td>
<td>0</td>
<td>12</td>
</tr>
<tr>
<td>5</td>
<td>436</td>
<td>0</td>
<td>295382</td>
<td>400840</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>1972</td>
<td>0</td>
<td>8198146</td>
<td>942926</td>
<td>0</td>
<td>22</td>
</tr>
<tr>
<td>7</td>
<td>1674</td>
<td>0</td>
<td>9722324</td>
<td>10477560</td>
<td>0</td>
<td>24</td>
</tr>
<tr>
<td>8</td>
<td>1034</td>
<td>0</td>
<td>6299844</td>
<td>10477560</td>
<td>0</td>
<td>24</td>
</tr>
<tr>
<td>9</td>
<td>271</td>
<td>0</td>
<td>571236</td>
<td>816936</td>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td>10</td>
<td>1378</td>
<td>0</td>
<td>6556248</td>
<td>7333340</td>
<td>0</td>
<td>23</td>
</tr>
</tbody>
</table>
The output in Figure 8.30 shows that several of the rollback segments are experiencing wrapping. The greatest amount of wrapping occurred in rollback segments 7 and 8, which each had 24 wraps occur between the times UTLBSTAT.SQL and UTLESTAT.SQL was executed.

**Dynamic Extent Allocation of Rollback Segments**

When a transaction’s before-image exceeds the size of the extent to which it has been allocated, but cannot wrap to the next extent because there are still active transactions in that extent, the rollback segment will add a new extent and wrap into that extent instead. This dynamic extent allocation incurs an I/O cost that should be avoided if possible. You can examine the V$ROLLSTAT and V$SYSTEM_EVENT dynamic performance views and the output from STATSPACK to determine whether dynamic extent allocation is occurring in your rollback segments.

**Using V$SYSTEM_EVENT to Measure Dynamic Undo Segment Extent Allocation**

The statistic undo segment extension in the V$SYSTEM_EVENT dynamic performance view records how long user Server Processes had to wait for rollback segments to add extents to handle transaction processing:

```sql
SQL> SELECT event, total_waits, 2       time_waited, average_wait 3 FROM v$system_event 4 WHERE event like '%undo%';
```

<table>
<thead>
<tr>
<th>EVENT</th>
<th>TOTAL_WAIT</th>
<th>TIME_WAITED</th>
<th>AVERAGE_WAIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>undo segment extension</td>
<td>28070</td>
<td>18</td>
<td>0.000641254</td>
</tr>
</tbody>
</table>

The output from this query indicates that application users waited 18 times for rollback segments to add extents in order to store their before images.

High or steadily increasing values for these wait statistics indicate that you either have too few rollback segments, too small rollback segments, or both.
Using `V$ROLLSTAT` to Measure Dynamic Undo Segment Extent Allocation

The `V$ROLLSTAT` view contains a column called `EXTENDS`. This column indicates how often the rollback segment was forced to add extents to support database transaction activity:

```
SQL> SELECT n.name, s.usn, s.extends
2   FROM v$rollname n, v$rollstat s
3   WHERE n.usn = s.usn;
```

<table>
<thead>
<tr>
<th>NAME</th>
<th>USN</th>
<th>EXTENDS</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSTEM</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>RBS01</td>
<td>1</td>
<td>55</td>
</tr>
<tr>
<td>RBS02</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>RBS03</td>
<td>3</td>
<td>12</td>
</tr>
<tr>
<td>RBS04</td>
<td>4</td>
<td>16</td>
</tr>
<tr>
<td>RBS05</td>
<td>5</td>
<td>8</td>
</tr>
<tr>
<td>RBSBIG</td>
<td>6</td>
<td>52</td>
</tr>
</tbody>
</table>

The output from the above query shows that several of the rollback segments had to dynamically allocate additional extents during application processing. The hardest hit were the RBS01 (55 extents added) and RBSBIG (52 extents added) undo segments.

If your rollback segments are frequently adding extents to support the transaction activity in your database, then the rollback segments are probably too small.

Using `STATSPACK` to Measure Dynamic Undo Segment Extent Allocation

Statistics on dynamic extent allocation for undo segments can also be found in the output from the `STATSPACK` utility. Figure 8.31 shows sample `STATSPACK` output that includes these statistics.
This output shows that, on 12 occasions, undo segments 17 and 18 had to add additional extents (denoted by Extends column) in order to hold the complete before-image data for the transactions assigned to them.

Using OEM Performance Manager to Measure Undo Segments

The Performance Manager component of the Oracle Enterprise Manager Diagnostics Pack includes several graphical representations of undo segment performance. These graphical displays can be used to monitor the same undo segment performance areas that were examined in the output from V$ dynamic performance views, STATSPACK and REPORT.TXT in the previous sections. Figure 8.32 shows a sample Performance Manager screen that displays rollback segment hit ratios.

FIGURE 8.32 Performance Manager Undo Segment Hit Ratios
The output shown in Figure 8.32 indicates that all the undo segments are achieving very high (nearly 100 percent) hit ratios. This means that application users are essentially able to access the undo segments whenever they request them, without waiting. Wait statistics for undo segments can be further examined using the Performance Manager GUI tool shown in Figure 8.33.

**Figure 8.33** Performance Manager undo wait statistics

The output shown in Figure 8.33 indicates that nearly 100 percent of the time, application users are not experiencing waits when requesting access to rollback segments. This is an indicator of good undo segment performance.

### Improving Rollback Segment I/O

After measuring the performance of your rollback segments, you may decide to tune them for improved performance. In general your undo segment tuning goals are to eliminate contention for rollback segments, try to minimize rollback segment extending and wrapping, avoid running out of space in undo segments, and always have rollback data needed for constructing read-consistent images for application users. In order to achieve these objectives, consider these four tuning categories:

- Add more rollback segments
- Make the existing rollback segments bigger
- Explicitly manage rollback segments for large transactions
- Minimize the need for rollback space
- Use automatic undo management features

The tricky question becomes how much bigger to make the rollback segments and how many more rollback segments to create.

While an exact determination of the proper combination of these techniques can be elusive, several useful guidelines are mentioned in the following sections.

**Oracle Objectives**

- Reconfigure and monitor rollback segments
- Define the number and sizes of rollback segments

### Add More Undo Segments to Improve Performance

The easiest way to minimize contention for both rollback segment headers and rollback segment extent blocks is to make more of those two resources available. To add these resources, you have to add more rollback segments to the database. To further enhance I/O performance, any additional rollback segments that may be added can be created in a new tablespace, separate from the existing rollback segments.

Oracle recommends 1 rollback segment for every four concurrent transactions, for up to a maximum of 20 rollback segments.

### Make Undo Segments Bigger to Improve Performance

Not only do you need to have enough rollback segments available in the database, but you must also make sure that they are large enough to handle the before-images that will be placed in them. The optimal size of rollback segments varies with the type of system (i.e. OLTP vs. DSS) and the types of transactions being performed in the database. Table 8.9 compares the relative costs of each type of DML command.
If the table the DML is being performed on also has an associated index or indexes, the demands on the rollback segment will be even greater since the index before-images must also be maintained.

To more accurately determine how much rollback space your users need, you can monitor how much they are currently using via the V$TRANSACTION, V$SESSION, and V$ROLLSTAT dynamic performance views.

**Using V$TRANSACTION and V$SESSION to Monitor Rollback Segment Usage**

By joining the V$SESSION and V$TRANSACTION views you can see how much space each session is using in the database rollback segments:

```
SQL> SELECT s.osuser, s.username, t.used_ublk
   2  FROM v$session s, v$transaction t
   3  WHERE s.taddr = t.addr;
```

<table>
<thead>
<tr>
<th>OSUSER</th>
<th>USERNAME</th>
<th>USED_UBLK</th>
</tr>
</thead>
<tbody>
<tr>
<td>rbron</td>
<td>SYSTEM</td>
<td>2</td>
</tr>
<tr>
<td>rbron</td>
<td>APPS</td>
<td>280</td>
</tr>
<tr>
<td>jschwarz</td>
<td>APPS</td>
<td>16</td>
</tr>
<tr>
<td>jjohnson</td>
<td>APPS</td>
<td>12</td>
</tr>
<tr>
<td>tgastel</td>
<td>APPS</td>
<td>8</td>
</tr>
</tbody>
</table>

This output shows the amount of rollback activity currently being generated by each user at the moment the query was issued. For example, user

---

**Table 8.9** Comparison of Rollback Cost for DML Operations

<table>
<thead>
<tr>
<th>Command</th>
<th>Entry Stored in Rollback Segment</th>
<th>Relative Rollback Segment Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>INSERT</td>
<td>The ROWID of the inserted row</td>
<td>Low</td>
</tr>
<tr>
<td>UPDATE</td>
<td>The column data for the changed columns</td>
<td>Medium</td>
</tr>
<tr>
<td>DELETE</td>
<td>The entire deleted row</td>
<td>High</td>
</tr>
</tbody>
</table>
rbron’s transaction is generating 290 blocks worth of undo information. If the Oracle block size is 8K, this means that rbron is generating just over 2MB of rollback data.

**Using V$ROLLSTAT to Monitor Rollback Segment Usage**

If you have a large transaction that seems to be having trouble with rollback segments, you may have difficulty using the above query on V$SESSION and V$TRANSACTION to track its rollback segment usage. In these cases, it is usually easier to use the V$ROLLSTAT view to determine how much rollback segment space the transaction needs. The procedure to do so is outlined here:

1. Take all but one rollback segment offline:
   
   ```sql
   SQL> ALTER ROLLBACK SEGMENT rbs01 OFFLINE;
   SQL> ALTER ROLLBACK SEGMENT rbs02 OFFLINE;
   SQL> ALTER ROLLBACK SEGMENT rbs03 OFFLINE;
   SQL> ALTER ROLLBACK SEGMENT rbs04 OFFLINE;
   SQL> ALTER ROLLBACK SEGMENT rbs05 OFFLINE;
   ```

2. Query the V$ROLLSTAT and V$ROLLNAME views to determine how many bytes of before-image data have been written to the rollback segment so far. This statistic is contained in the WRITES column of V$ROLLSTAT:
   
   ```sql
   SQL> SELECT n.name, s.usn, s.writes
   2   FROM v$rollname n, v$rollstat s
   3   WHERE n.usn = s.usn
   4   AND name != 'SYSTEM';
   ```

<table>
<thead>
<tr>
<th>NAME</th>
<th>USN</th>
<th>WRITES</th>
</tr>
</thead>
<tbody>
<tr>
<td>RBS06</td>
<td>6</td>
<td>4498</td>
</tr>
</tbody>
</table>

   This output indicates that thus far, 4,498 bytes worth of undo data have been written to rollback segment number six.

3. Next, run the transaction that is experiencing rollback segment problems:
   
   ```sql
   SQL> DELETE FROM sales_history
   2   WHERE sale_date < '01-JAN-95';
   ```
4. Now, rerun the query on V$ROLLSTAT and V$ROLLNAME views to see how many bytes have been written to the rollback segment now:

```
SQL> SELECT n.name, s.usn, s.writes
FROM v$rollname n, v$rollstat s
WHERE n.usn = s.usn
AND name != 'SYSTEM';
```

<table>
<thead>
<tr>
<th>NAME</th>
<th>USN</th>
<th>WRITES</th>
</tr>
</thead>
<tbody>
<tr>
<td>RBS06</td>
<td>6</td>
<td>6198468</td>
</tr>
</tbody>
</table>

Since the WRITES column records the number of bytes that have been written to each rollback segment, the difference between the value for WRITES in Step 2 and the value for WRITES in Step 4 will indicate the size of the transaction’s rollback requirements. In this example: 6,198,468 – 4,498 = 6,193,970 bytes. This means that the overall size of your rollback segments will need to be at least 6MB if they are to be expected to handle this transaction. However, this method can be wasteful of disk space if each rollback segment is sized for your largest transaction. The next section, “Explicitly Managing Rollback Segments,” discusses how to alleviate this problem.

### Real World Scenario

**More and Bigger: The Special Case of **SNAPSHOT TOO OLD**

The bane of many a DBA is the unpredictable occurrence of the ORA-01555 SNAPSHOT TOO OLD (rollback segment too small) error. These errors can be hard to resolve because they don’t always occur in a consistent manner. The following scenario illustrates how a SNAPSHOT TOO OLD error might occur and why these errors are hard to tune away.

- **11:30 A.M.:** Malka updates a customer’s unpaid balance, thereby writing the before-image of the data to the database’s rollback segment. She is then interrupted by a phone call, leaving her changes temporarily uncommitted.
- **11:32 A.M.:** Jennifer starts a job that generates a report showing all unpaid balances on customer accounts, including the customer that Malka is in the process of updating. For read consistency, the Oracle
When I create rollback segments on new systems, I use standard sizing values of INITIAL 512K, NEXT 512K, and a MINEXTENTS of 20. This formula will result in a 10MB rollback segment. These values tend to prevent most rollback segment problems, but should be monitored nonetheless to determine whether they are too big or too small.
Explicitly Managing Rollback Segments to Improve Performance

Very large transactions, like those associated with batch processing runs, frequently require large rollback segments, which are not necessarily appropriate for regular OLTP transactions.

Oracle Objective

In these cases, it is best to create one or two large rollback segments and dedicate them to this purpose. However, since the Oracle Server normally assigns transactions to rollback segments automatically, you will need to add a few extra steps to your batch processing in order to utilize this technique:

1. Create a new, large rollback segment for storing the before-images for large batch jobs:
   
   ```sql
   SQL> CREATE PRIVATE ROLLBACK SEGMENT rbsbatch
       2  STORAGE (INITIAL 10M NEXT 10M)
       3  TABLESPACE rbs;
   ```

   Right after creation, the new rollback segment will be in an offline state. Leave the rollback segment offline until it is time to do the batch processing.

2. Just before starting the batch processing jobs, put the rollback segment online:

   ```sql
   SQL> ALTER ROLLBACK SEGMENT rbsbatch ONLINE;
   ```

3. Next, assign the batch job to this rollback segment using the SQL command SET TRANSACTION or the DBMS_TRANSACTION stored procedure:

   ```sql
   SQL> SET TRANSACTION USE ROLLBACK SEGMENT rbsbatch;
   ```

   or

   ```sql
   SQL> EXECUTE DBMS_TRANSACTION.USE_ROLLBACK_SEGMENT('rbsbatch');
   ```

4. Now, start the batch processing job:

   ```sql
   SQL> EXECUTE posting_job;
   ```
5. Next, from a database session other than the one that is executing the batch job, immediately take the large rollback segment offline:

   SQL> ALTER ROLLBACK SEGMENT rbsbatch OFFLINE;

   The rollback segment will not actually go offline until the batch transaction is complete. However, because it has been marked to go offline in the data dictionary, no other transactions will be assigned to it. Manually assigning the transaction to its own rollback segment will help ensure that the transaction will find adequate undo space to hold the transaction’s before-image.

**WARNING**

Any commit that occurs within the batch job will cause the large rollback segment to go offline and become unavailable. Therefore, you must bring the rollback segment back online and reissue the SET TRANSACTION or DBMS_TRANSACTION command after each commit to keep the batch job from using some other rollback segment.

**Minimizing Rollback Segment Activity to Improve Performance**

Another way to minimize performance problems related to rollback segments is to try and minimize the number and size of entries that are written to the rollback segments. This goal can be accomplished by performing frequent commits to minimize large rollback entries, using the COMMIT=Y option when performing database imports, forgoing the use of the CONSISTENT option when using the database EXPORT utility, and setting an appropriate COMMIT value when using the SQL*Loader utility.

**NOTE**

See Oracle9i Database Utilities, Release 1 (9.0.1) (Part Number A90192-01) for more information on the options for the EXPORT, IMPORT, and SQL*Loader utilities.

**Using Automatic Undo Management Features to Improve Performance**

Oracle9i offers a new feature, Automatic Undo Management (AUM), which relieves the DBA of many of the burdens of manual rollback segment undo (RBU) management described in the previous sections.
Automatic Undo Management (also known as System-managed Undo) is designed to minimize undo segment performance problems by dynamically managing the size and number of undo segments. This concept is similar to the dynamic management of sort segments in temporary tablespaces. With AUM, dedicated tablespaces, called Undo Tablespaces, are created to hold before-image data. The number and size of these undo segments is automatically managed by Oracle based on the demands of the system. The following sections describe techniques for using this new feature to improve undo segment I/O.

**Oracle Objective**

**Explain the concept of automatic undo management**

**Configuring Automatic Undo Management**

In order to utilize AUM in your database, you must first set four init.ora parameters to appropriate values. Descriptions of these parameters are included below.

**UNDO_MANAGEMENT** Used to indicate whether automatic (AUTO) or manual (MANUAL) undo management is being used in the database. The default is MANUAL.

**UNDO_TABLESPACE** Specifies the name of the tablespace where the AUM undo segments are to be stored. The default value is the first undo tablespace found during database startup.

If no undo tablespace exists, the SYSTEM rollback segment will be used to store undo data.

**UNDO_RETENTION** Specifies (in seconds) how long to retain undo information after the transaction that generated it, commits. The default value is 900 seconds. This parameter is designed to help minimize Snapshot Too Old errors caused when a transaction needs overwritten before-image data for read consistency.

Undo data can only be retained up to the time specified by UNDO_RETENTION when adequate space has been allocated to the undo tablespace.
UNDO_SUPPRESS_ERRORS  Used to suppress errors that would otherwise occur when commands that are appropriate in RBU mode (e.g., SET TRANSACTION USE ROLLBACK SEGMENT ...) are used in AUM mode.

Once these parameters are configured, you must next create an undo tablespace to hold the system-managed undo segments.

Creating an Undo Tablespace

Once the UNDO_MANAGEMENT init.ora parameter has been set to AUTO, you must create a tablespace to hold the system-managed undo segments. The syntax for creating a AUM tablespace is shown here:

```sql
SQL> CREATE UNDO TABLESPACE undo
2  DATAFILE 'd:\oradata\PROD\undo01.dbf' SIZE 500M
3  AUTOEXTEND ON
4  NEXT 5M MAXSIZE 2000M;
```

Once the UNDO tablespace has been created using the above syntax, the parameter UNDO_TABLESPACE=UNDO must be added to init.ora, and the instance restarted before the new tablespace will be used for AUM.

You cannot specify the default initial and next extent size for an UNDO tablespace. These values will be system-generated.

The undo tablespace can also be defined at database creation within the CREATE DATABASE statement itself. Figure 8.34 shows an example of this syntax.

**FIGURE 8.34**  Undo tablespace specified at database creation

```sql
SQL> CREATE DATABASE PROD
2  MAXDATAFILES 500
3  MAXLOGMEMBERS 4
4  DATAFILE 'd:\oradata\PROD\system01.dbf' SIZE 250M
5  UNDO TABLESPACE rds DATAFILE 'f:\oradata\PROD\rds01.dbf' SIZE 500M
6  LOGFILE GROUP 1 ('d:\oradata\PROD\log1a.ord', 'd:\oradata\PROD\log1b.ord') size 5M,
7  GROUP 2 ('d:\oradata\PROD\log2a.ord', 'd:\oradata\PROD\log2b.ord') size 5M;
```
Figure 8.34 shows how a 500MB undo tablespace called RBS could be specified during the creation of a new database called PROD. If the UNDO_MANAGEMENT parameter is set to AUTO and no undo tablespace is specified at database creation, Oracle will create a tablespace called SYS_UNDOTBS to store the undo segment. The Datafile for this tablespace will be created in $ORACLE_HOME/dbs on Unix systems and %ORACLE_HOME%/dbs on Windows 2000 systems.

Managing Undo Tablespace
While a database can have several undo tablespaces, only one active undo tablespace is allowed in the database at a time once automatic undo management is configured. However, if you need to switch the database from one system-managed undo tablespace to another you can do so using the following SQL command:

```
SQL> ALTER SYSTEM SET UNDO_TABLESPACE=undo;
```

Once the above command has been issued, any new transactions will write their before-image data to the UNDO tablespace. However, any transactions that began before the ALTER SYSTEM command was issued will store their before-image data in the previous undo tablespace until those transactions complete and the UNDO_RETENTION period has passed.

With these restrictions, an undo tablespace can also be dropped using the DROP TABLESPACE command if you find that it is no longer needed:

- The tablespace to be dropped must not be the active undo tablespace
- There must be no active, uncommitted transactions storing before image data in the tablespace

Any transaction that needed the before-image data stored in a dropped undo tablespace will receive an error when that data is needed for read consistency.

Monitoring the Performance of System-Managed Undo
Once created, you can use the DBA_TABLESPACES and V$UNDOSTAT views to monitor system-managed undo tablespaces.

**USING DBA_TABLESPACE TO MANAGE UNDO TABLESPACES**
The CONTENTS column in the DBA_TABLESPACES data dictionary view will contain an entry of UNDO for each AUM tablespace in the database. The
following query shows that this database has three undo tablespaces, the default undo tablespace UNDOTBS, and two other undo tablespaces, UNDO and RBS:

```
SQL> SELECT tablespace_name, contents
2  FROM dba_tablespaces;
```

<table>
<thead>
<tr>
<th>TABLESPACE_NAME</th>
<th>CONTENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSTEM</td>
<td>PERMANENT</td>
</tr>
<tr>
<td>UNDOTBS</td>
<td>UNDO</td>
</tr>
<tr>
<td>APPL_IDX</td>
<td>PERMANENT</td>
</tr>
<tr>
<td>APPL_TAB</td>
<td>PERMANENT</td>
</tr>
<tr>
<td>TEMP</td>
<td>TEMPORARY</td>
</tr>
<tr>
<td>TOOLS</td>
<td>PERMANENT</td>
</tr>
<tr>
<td>USERS</td>
<td>PERMANENT</td>
</tr>
<tr>
<td>OEM_REPOSITORY</td>
<td>PERMANENT</td>
</tr>
<tr>
<td>UNDO</td>
<td>UNDO</td>
</tr>
<tr>
<td>RBS</td>
<td>UNDO</td>
</tr>
</tbody>
</table>

**USING V$UNDOSTAT TO MANAGE UNDO TABLESPACES**

The V$UNDOSTAT dynamic performance view can be used to monitor the usage of the system-managed undo tablespace. This view contains historical statistics, presented as a histogram. Each row in the V$UNDOSTAT view stores undo statistics from a previous 10-minute interval. Table 8.10 describes selected columns from this view.

**Table 8.10** Descriptions of Selected V$UNDOSTAT Columns

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BEGIN_TIME</td>
<td>Time and date when the undo statistics monitoring period began.</td>
</tr>
<tr>
<td>END_TIME</td>
<td>Time and date when the undo statistics monitoring period ended.</td>
</tr>
<tr>
<td>UNDOTSN</td>
<td>The ID of the tablespace where the undo activity took place. Can be joined to V$TABLESPACE on the TS# column in order to determine the name of the tablespace.</td>
</tr>
</tbody>
</table>
Tuning Rollback Segment I/O

Figure 8.35 shows a sample query on V$UNDOSTAT.

**FIGURE 8.35** Sample query on V$UNDOSTAT

```
BEGIN
  SELECT TO_CHAR(u.begin_time, 'DD-MON-YY HH:MI AM') "BEGIN",
         TO_CHAR(u.end_time, 'DD-MON-YY HH:MI AM') "END",
         t.name "TABLESPACE",
         u.undoblocks "BLOCKS USED",
         u.txcnt "TRANSACTIONS",
         u.maxqlen "LONGEST QUERY",
         u.unxpblkrecnt "EXPIRED BLOCKS"
  FROM v$undostat u, v$tablespace t
  WHERE u.tablespace = t.tablespace
END;
```

Using the output in Figure 8.35 it is possible to determine whether the undo tablespace is properly tuned. Examples of performance indicators are discussed below.

**BLOCKS USED** Column  Derived from the UNDOBLKS column, the BLOCKS USED column in Figure 8.34 shows that a maximum of 8,232 blocks were used to store before-image data in any 10-minute period since instance startup. If the Oracle block size is 8K, this would indicate that the

**TABLE 8.10** Descriptions of Selected V$UNDOSTAT Columns (continued)

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNDOBLKS</td>
<td>The number of undo blocks used to store before-image data between the begin and end time.</td>
</tr>
<tr>
<td>TXNCOUNT</td>
<td>The total number of transactions executed between the begin and end time.</td>
</tr>
<tr>
<td>MAXQUERYLEN</td>
<td>The time (in seconds) that the longest query took to execute between the begin and end time. Use this value to determine the appropriate value for UNDO_RETENTION.</td>
</tr>
<tr>
<td>UNXPBLKREUCNT</td>
<td>The number of undo blocks that were needed to maintain read consistency for another transaction, but which were reused to hold another transaction’s before-image instead. Use this value to measure the likelihood of Snapshot Too Old errors occurring in the database.</td>
</tr>
</tbody>
</table>
undo tablespace should be at least 64MB (i.e., 8K block size * 8,232 blocks used = 674,365,444 bytes = 64MB) in order to hold the before-image data needed by application users.

**TRANSACTIONS Column** Derived from the TXNCOUNT column, the TRANSACTIONS column in Figure 8.34 shows that a maximum of 2,212 transactions have occurred in any 10-minute period since instance startup. This information is useful because it helps you be sure that the statistics shown in V$UNDOSTAT were gathered when representative application activity was taking place.

**LONGEST QUERY Column** Derived from the MAXQUERYLEN column, the LONGEST QUERY column indicates the longest-running query since instance startup ran for 221 seconds. This means that the value for UNDOibern should be set to at least 221 seconds to help minimize Snapshot Too Old errors associated with missing undo data.

**EXPIRED BLOCKS Column** Derived from the UNXPBLKREUNCNT column, the EXPIRED BLOCKS column indicates that 12 undo blocks were reused (expired) before the transactions that were “interested” in those blocks completed. A frequent occurrence of this statistic also indicates potential for Snapshot Too Old errors.

**Summary**

Oracle background processes and user transaction activity both cause I/O to occur during normal operation. Although proper tuning of the SGA can greatly minimize unnecessary I/O, you should still monitor and tune the I/O-specific operations in the database. Effective I/O tuning includes managing Datafile I/O, write activities of the DBW0 block-level I/O, sort activities, and rollback segments.

Datafile I/O statistics can be found in the V$FILESTAT and V$DATAFILE dynamic performance views as well as the output from STATSPACK and REPORT.TXT, and the OEM Performance Tuning GUI tool. Datafile and tablespace I/O is generally tuned by separating application data from data dictionary objects, segregating application tables and indexes, locating Datafiles on several different physical devices via striping, and using locally managed tablespaces.

DBW0 performance is tuned by monitoring its performance and starting Database Writer I/O slaves or additional Database Writer processes as needed.
Block-level I/O is dependent on several factors including block size, extent size, and update frequency. Common I/O problems related to Oracle blocks include row migration and row chaining.

I/O occurs any time a user’s Server Process performs a sort that is larger than the value specified by `SORT_AREA_SIZE` in `init.ora`. The use of true temporary tablespaces and tempfiles helps improve the performance of disk sorts.

Rollback segments represent a special challenge to database tuners (DBAs). Because every user must be able to get a read-consistent view of the database and be able to write before-image data for their transactions, rollback segments have a tendency to grow dynamically and be the source of heavy I/O. Minimizing wrapping and dynamic extension of rollback segments are two key I/O tuning goals. Utilizing automatic undo management through the use of undo tablespaces can help improve undo segment performance by allowing Oracle to dynamically manage the size and number of undo segments in the database.

### Exam Essentials

**Understand how to measure Datafile and tablespace I/O.** Know which dynamic performance views, STATSPACK output and REPORT.TXT data can be used to determine the effectiveness of Datafile and tablespace I/O. Be able to interpret the contents of these sources to determine if performance problems exist. Understand how to determine if tablespaces are configured appropriately.

**Be able to describe techniques to improve Datafile and tablespace I/O.** Know how Datafile storage options like manual striping, RAID storage, and asynchronous devices can be used to enhance I/O performance.

**Know how Oracle manages segment storage and I/O.** Be able to describe the relationship between blocks and extents. Know the role that `PCTFREE`, `PCTUSED`, and the High Water Mark play in segment block usage. Understand the causes of row chaining and migration and how to detect and correct these problems.
Understand how to measure and tune sorting-related I/Os. Know which SQL operations cause sorting to occur. Understand what steps can be taken at the statement level and instance level to minimize sorting, and what techniques can be used to perform sorting as quickly as possible when it does occur. Be able to describe how the use of temporary tablespaces and tempfiles improves sorting performance.

Be able to explain how to measure and tune rollback segment I/O. Understand which data dictionary and dynamic performance views contain information about undo segment performance. Know how to use the output from STATSPACK and REPORT.TXT to analyze rollback segment I/O. Be able to describe how the use of automatic undo management and undo tablespaces can be used to improve rollback segment performance.

**Key Terms**

Before you take the exam, be certain you are familiar with the following terms:

- asynchronous
- rollback
- Automatic Undo Management
- row migration
- chained row
- segment header
- Datafiles
- sorted
- extent
- striping
- extent map
- synchronous
- Free List
- System-managed Undo
- High Water Mark
- tablespace
- Locally Managed Tablespaces
- tempfiles
- Primary Block Size
- undo
- RAID
- undo segments
Review Questions

1. Which of the following dynamic performance views contain information related to tuning I/O? (Choose all that apply.)
   A. V$SYSSTAT
   B. V$LIBRARYCACHE
   C. V$SYSTEM_EVENT
   D. V$FILESTAT

2. Which of the following is not a possible source for I/O related performance issues?
   A. Datafiles
   B. Oracle blocks
   C. Control files
   D. Redo logs

3. The values for MINIOTIM and AVGIOTIM in V$FILESTAT are expressed in which unit?
   A. Minutes
   B. Seconds
   C. 100ths of a second
   D. 1000ths of a second

4. Suppose you have determined that the EMP table and its index on LAST_NAME are frequently being used in application processing. What would be one way to try and improve the I/O of this operation?
   A. Make sure the table and its index are in the same tablespace.
   B. Make sure the table and its index are on the same physical device.
   C. Ask users to commit their work more often.
   D. Separate the table and its index onto separate tablespaces and physical devices.
5. Separating an application’s accounting tables from its manufacturing tables is an example of which of the following options?
   A. Segregating segments by size.
   B. Segregating segments by functional area.
   C. Segregating segments by growth potential.
   D. None of the above is correct.

6. Which of the following is not an example of a Datafile striping technique?
   A. Placing a single Datafile on a RAID device
   B. Creating a tablespace with several Datafiles and placing the Datafiles on one device
   C. Creating a tablespace with several Datafiles and placing each Datafile on a separate device
   D. Creating a tablespace with several Datafiles and placing the Datafiles on a RAID device

7. Why do locally managed tablespaces offer a performance advantage?
   A. Their storage management is performed in memory.
   B. Their extents are larger than those of a regular tablespace.
   C. They do not use the data dictionary to manage their storage allocations.
   D. They can be of the type temporary or permanent.

8. Which of the following describes the advantage of asynchronous over synchronous I/O devices?
   A. They do not rely on the OS layer to perform their write operations.
   B. They can use disks larger than 1GB.
   C. They can be backed up to tape.
   D. They generally require more skill to create and manage.
9. How do the actions of a Database Writer I/O slave and the DBW0 background process differ?
   A. I/O slaves are only used during recovery.
   B. DBW0 only performs write activities, while the I/O slaves also manage the Database Buffer Cache LRU List.
   C. I/O slaves can only read from Datafiles, not write to them.
   D. I/O slaves only perform write activities, while DBW0 also manages the Database Buffer Cache LRU List.

10. What statistic in V$SYSSTAT indicates that DBW0 may not be writing the contents of the Database Buffer Cache to disk fast enough?
    A. redo log space requests
    B. buffer busy wait
    C. db file parallel write
    D. busy buffer wait

11. Which two init.ora parameters are used to tune the performance of Database Writer?
    A. DB_BLOCK_WRITERS
    B. DBWR_IO_SLAVES
    C. DB_WRITER_PROCESSES
    D. DB_WRITERS

12. What happens to a database block once it has been filled to PCTFREE?
    A. The rows in the block are migrated to a new block.
    B. Any new rows will be chained to another block.
    C. The block goes onto the Used List.
    D. The block comes off the Free List.
13. A row is updated so that it fills all the available space in the block where the row resides. Any subsequent updates to rows in that block will cause what reaction?
   A. Row chaining
   B. Row migration
   C. Row compression
   D. Row corruption

14. Which of the following is identified using the ANALYZE TABLE LIST CHAINED ROWS command? Select all that apply.
   A. Row migration
   B. Row chaining
   C. Row corruption
   D. Row length

15. During a full table scan, the user’s Server Process reads all the table’s blocks up to what point?
   A. Last extent
   B. PCTFREE value
   C. High Water Mark
   D. User’s tablespace quota

16. Which of the following is the primary determinant of whether a sort will happen in memory or to disk?
   A. The amount of memory in the user’s PC
   B. The amount of memory in the server
   C. The amount of server memory reserved for sorting by the SORT_AREA_SIZE init.ora parameter
   D. The number of tables involved in the query causing the sort
17. Following a sort, the Oracle Server shrinks the sort area assigned to each user’s PGA to the value specified by:
   A. SORT_AREA_SIZE
   B. SORT_AREA_RETAINED_SIZE
   C. SORT_AREA_MIN_SIZE
   D. SORT_AREA_MAX_SIZE

18. When a rollback segment continues writing its before-image from one extent to the next adjacent extent in a rollback segment, this is referred to as which of the following?
   A. Wrapping
   B. Extending
   C. Snapshot Too Old
   D. None of the above

19. Which of the following might be possible solutions for resolving an ORA-01555 SNAPSHOT TOO OLD error? Select all that apply.
   A. Add more rollback segments.
   B. Make the existing rollback segments larger.
   C. Avoid running long queries against tables that are frequently updated.
   D. All of the above are possible solutions.

20. What init.ora parameter determines the maximum number of database blocks that are read by a user’s Server Process whenever a full table scan read operation is performed?
   A. DB_FILE_MAX_READ_COUNT
   B. DB_FILE_MULTIBLOCK_READ_COUNT
   C. DB_FILE_MULTIBLOCK_MAX
   D. DB_FILE_MULTIBLOCK_READ_MAX
21. You want to start using the Automatic Undo Management feature of Oracle9i. What parameter needs to be set to allow the use of this feature?
   A. UNDO_MANAGEMENT=AUTO
   B. MANAGED_UNDO=ON
   C. UNDO_MANAGEMENT=TRUE
   D. MANAGED_UNDO=TRUE

22. Choose two new Oracle9i parameters that are designed to help the DBA manage sorts that occur in memory more effectively:
   A. WORKAREA_SIZE_POLICY
   B. PGA_AGGREGATE_TARGET
   C. SORT_AREA_SIZE
   D. SORT_AREA_RETAINED_SIZE

23. In Oracle9i, which statement is false about block size?
   A. It can be set at the tablespace level.
   B. It can be set at the database level.
   C. Settings at the database level take precedence over settings at the tablespace level.
   D. Settings at the tablespace level take precedence over settings at the database level.

24. You start a query that is using the parallel query operation and want to monitor the progress of the query. What dynamic performance view can be used to monitor this query?
   A. V$SGA
   B. V$QUERYSTAT
   C. V$OPERATION
   D. V$SESSION_LONGOPS
25. Choose an appropriate PCTFREE setting for a table whose rows will never change in size?

A. 0
B. 40
C. 10
D. 100

26. Which one of the following commands would not move the High Water Mark for the EMP table?

A. TRUNCATE TABLE EMP;
B. ALTER TABLE EMP MOVE TABLESPACE USER_DATA;
C. ALTER TABLE EMP DEALLOCATE UNUSED;
D. DROP TABLE EMP;
Answers to Review Questions

1. A, C, D. The V$LIBRARYCACHE view contains information about the Shared Pool, not I/O.

2. C. Very little I/O occurs against the database control files. They are only briefly updated at a checkpoint event by the CKPT background process.

3. C. The columns MAXIOWTM and MAXIORTM in V$FILESTAT are also expressed in 100ths of a second.

4. D. Separating the table from its indexes will help improve the performance of OLTP transactions that access both segments during processing.

5. B. Separating segments by functional area helps to balance the I/O between application components that are used by different user groups.

6. B. Effective Datafile striping requires that the Datafile or Datafiles for a tablespace be placed on multiple devices, either via a RAID or manual placement on physical disks.

7. C. Instead of the data dictionary, locally managed tablespaces use a bitmap in the file header of each Datafile to store space allocation information.

8. A. Asynchronous, or raw, devices are written to directly by Oracle without waiting for a signal from the OS as is required by synchronous, or cooked, devices.

9. D. I/O slaves are only designed to assist Database Writer with its write activities. They do not move blocks from the LRU List to the Dirty List.

10. A. If DBW0 is not writing the contents of the Database Buffer Cache to disk fast enough at a database checkpoint, LGWR will not be able to empty the contents of the Redo Log Buffer, causing the redo log space requests event to occur.

11. B, C. DBWR_IO_SLAVES is used to start I/O slaves. DB_WRITER_PROCESSES is used to start additional Database Writer processes.

12. D. Blocks that have been filled to their PCTFREE value are removed from the Free List and can no longer accept new inserts. The blocks go back on the Free List once the block’s storage drops below PCTUSED.
13. B. Rows that are located in a block whose PCTFREE space is completely consumed are migrated to another block when they are updated.

14. A, B. The LIST CHAINED ROWS phrase identifies chained and/or migrated rows by populating the CHAIN_CNT column in DBA_TABLES with a numeric value.

15. C. All table blocks up to the High Water Mark are read during full table scans, whether or not those blocks contain data.

16. C. The SORT_AREA_SIZE parameter allows you to specify how much memory to dedicate to sorting in each user’s PGA. If the sort is larger than this value, the sort will be done to disk in several chunks.

17. B. Following a sort, the amount of sort space allocated to a user’s PGA is decreased to the value specified by SORT_AREA_RETAINED_SIZE. These two values are equal by default.

18. A. A wrap occurs whenever a user’s before-image fills the extent they were assigned to and then moves to store the remaining before-image in the adjacent extent. If the adjacent extent is in use, the rollback segment will add an additional extent.

19. D. The ORA-01555 error is associated with long-running queries that cannot find the read consistent view of the database that they require. This error can be remedied by making sure that before-images are maintained as long as possible after they are released by creating larger and more numerous rollback segments.

20. B. The DB_FILE_MULTIBLOCK_READ_COUNT parameter determines the maximum number of blocks during a full table scan. This parameter can enhance I/O performance if your application uses many full table scans.

21. A. The new Automatic Undo Management feature of Oracle9i is initialized using the parameter UNDO_MANAGEMENT=AUTO. This allows Oracle to manage the rollback areas for you.

22. A, B. Oracle has introduced two new parameters to allow you to manage sort space more effectively: WORKAREA_SIZE_POLICY and PGA_AGGREGATE_TARGET. Both of these parameters are designed to specify thresholds on the amount of sort space allocated. This can help to prevent a process from compromising system performance by grabbing more memory than is desired and by setting an upper boundary on available sort space for all sessions.
23. C. Oracle9i now allows you to set varying block sizes on a tablespace-by-tablespace basis. This gives great flexibility in how you can lay out your database. Settings at the tablespace level take precedence over settings at the database level. So if you want to have a different block size than the default block size, you would set this at the time you create the tablespace.

24. D. V$SESSION_LONGOPS can provide information on operations that run for a longer period of time (6 seconds in absolute time).

25. A. If rows in a table will never grow or shrink, then you could set the PCTFREE to 0. This means you would use the full capacity of the block for all inserted rows, thereby alleviating wasted space in the block.

26. C. When you use the DEALLOCATE UNUSED clause of the ALTER TABLE command, this de-allocates unused space above the High Water Mark. This does not move the high water mark.
Tuning Contention

ORACLE9i PERFORMANCE TUNING EXAM
OBJECTIVES COVERED IN THIS CHAPTER:

✓ Detect and resolve freelist contention
✓ Describe the purpose of latches
✓ Describe the different types of latch requests
✓ Diagnose contention for latches
✓ Tune the appropriate resources to minimize latch contention
✓ Define levels of locking
✓ Describe possible causes of contention
✓ Use Oracle utilities to detect lock contention
✓ Resolve contention in an emergency
✓ Prevent locking problems
✓ Recognize Oracle errors arising from deadlocks

Exam objectives are subject to change at any time without prior notice and at Oracle’s sole discretion. Please visit Oracle’s Certification website (http://www.oracle.com/education/certification/) for the current exam objectives listing.
This chapter deals with tuning resource contention. Contention for Oracle resources occurs any time an Oracle process tries to access an Oracle structure, but is unable to gain access to the structure because it is already in use by another process. Latches, Free Lists, and locking are all common sources of contention. Latch contention can be used as a secondary indicator of the performance of the Shared Pool, Database Buffer Cache, and Redo Log Buffer. Free List contention can cause DML operations on busy tables to perform poorly. Lock contention can have a huge performance impact if application users are frequently experiencing a complete halt in their processing due to overly restrictive locking. Each of these types of contention is examined in this chapter.

**Latch Contention**

Latches are used to protect access to Oracle’s memory structures. A *latch* is a specialized type of lock that is used to serialize access to a particular memory structure or serialize the execution of kernel code. Each latch protects a different structure or mechanism as indicated by the name of the latch. Only one process at a time may access a latch; processes are allowed to access a latch only when the latch is not already in use by another process. In this manner, Oracle makes sure that no two processes are accessing the same data structure simultaneously.

**Oracle Objectives**

- Describe the purpose of latches
- Describe the different types of latch requests
If a process needs a latch that is busy when the process requests it, the process will experience a wait. This wait behavior varies with the type of latch being accessed:

- If the latch is a *Willing-to-Wait* latch, the process requesting the latch will wait for a short period and then request the latch again, perhaps waiting several more times, until it successfully attains the requested latch.

- If the latch is an *Immediate* latch, the process requesting the latch continues to carry out other processing directives instead of waiting for the latch to become available.

The V$LATCH dynamic performance view is used to monitor the activity of both Willing-to-Wait and Immediate latches. Table 9.1 shows some of the columns found in V$LATCH that are important for monitoring latch contention.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAME</td>
<td>The name of the latch</td>
</tr>
<tr>
<td>GETS</td>
<td>The number of times a Willing-to-Wait latch was acquired without waiting</td>
</tr>
<tr>
<td>MISSES</td>
<td>The number of times a Willing-to-Wait latch was not acquired and a wait resulted</td>
</tr>
<tr>
<td>SLEEPS</td>
<td>The number of times a process had to wait before obtaining a Willing-to-Wait latch</td>
</tr>
<tr>
<td>IMMEDIATE_GETS</td>
<td>The number of times an Immediate latch was acquired without waiting</td>
</tr>
<tr>
<td>IMMEDIATE_MISSES</td>
<td>The number of times an Immediate latch was not acquired and a retry resulted</td>
</tr>
</tbody>
</table>

In terms of performance tuning, we would like to minimize the number of times processes find either type of latch unavailable at the time they request them.
Latch behavior differs on single and multiple CPU servers. On a single CPU server, a process requesting an in-use latch will release the CPU and sleep before trying the latch again. On multiple CPU servers, a process requesting an in-use latch will “spin” on the CPU a specific number of times before releasing the CPU and trying again. The number of spins the process will use is OS specific.

The following sections will show you how to detect latch contention if it is occurring in the instance.

**Measuring Latch Contention**

Several sources of information on the performance of latches are available. These sources include the V$LATCH dynamic performance view, the output from STATSPACK and REPORT.TXT, and the OEM Performance Manager. However, before examining any of these sources, you should determine if latch contention is even occurring in your instance using the V$SYSTEM_EVENT dynamic performance view. Latch contention can be detected by examining the V$SYSTEM_EVENT view for occurrences of the latch free event. The following query shows a query that can be used for this purpose:

```sql
SQL> SELECT event, total_waits, time_waited
2  FROM v$system_event
3  WHERE event = 'latch free';
```

<table>
<thead>
<tr>
<th>EVENT</th>
<th>TOTAL_WAITS</th>
<th>TIME_WAITED</th>
</tr>
</thead>
<tbody>
<tr>
<td>latch free</td>
<td>59</td>
<td>22</td>
</tr>
</tbody>
</table>

The query shows that 59 latch-related wait events have occurred in the instance since startup. The total time spent waiting for these latches was 22 hundredths of a second.

**Oracle Objectives**

- Diagnose contention for latches
- Tune the appropriate resources to minimize latch contention
Once latch contention has been identified, you must determine which of the over 200 latches are experiencing the contention. Fortunately, only some of these 200-plus possible latches are of importance to the DBA from a tuning perspective. Descriptions of these latches, and techniques for improving their wait activities, are shown below.

**Shared Pool Latch**  The Shared Pool latch is used to protect access to the Shared Pool’s memory structures. Frequent waits for access to this latch indicate that the Shared Pool should be examined for possible performance improvement using the techniques described in Chapter 4, “Tuning the Shared Pool.”

Using the Shared Server option without having the Large Pool configured can lead to high Shared Pool latch contention.

**Library Cache Latch**  Like the Shared Pool latch, frequent waits for the Library Cache latch also indicates a poorly tuned Shared Pool. Use the techniques in Chapter 4 to tune the Shared Pool’s Library Cache.

**Cache Buffers LRU Chain Latch**  The Cache Buffers LRU Chain latch is used to manage the blocks on the LRU List in the Database Buffer Cache. This latch is used when Database Writer writes dirty buffers to disk and when a user’s Server Process searches the LRU list for a free buffer during a disk read. Frequent waits for the Cache Buffer LRU Chain latch can be caused be traced to two possible sources:

- Inefficient application SQL that results in excessive full table scans or inefficient execution plans. See Chapter 3 “SQL Application Tuning and Design,” for techniques on improving the performance of application SQL.
- Database Writer is unable to keep up with write requests. See Chapter 8 “Tuning Disk I/O,” for techniques on tuning Database Writer I/O.

**Cache Buffers Chains Latch**  The Cache Buffers Chains latch is accessed by user Server Processes when they are attempting to locate a data buffer that is cached in the Database Buffer Cache. Waits for this latch indicate
Chapter 9 • Tuning Contention

that some cached blocks are probably being repeatedly searched for in the Buffer Cache. Online Transactional systems with large block sizes have a tendency to experience Cache Buffers Chains latch waits more that systems with smaller block sizes. See Chapter 8 for details on how database block size affects I/O.

Formerly, two Oracle8i init.ora parameters, DB_BLOCK_LRU_LATCHES and CPU_COUNT, were used to tune some aspects of Buffer Cache latch activity. These parameters have been deprecated in Oracle9i.

**Redo Allocation Latch**  The Redo Allocation latch is used to manage the space allocation in the Redo Log Buffer. Contention can occur for this latch if many users are trying to place redo entries in the Redo Log Buffer at the same time. Waits for the Redo Allocation latch can be minimized using the Redo Log Buffer tuning techniques described in Chapter 7, “Tuning Redo Mechanisms.”

**Redo Copy Latch**  Redo Copy latches are accessed by user Server Processes when they are copying their redo information into Redo Log Buffer. Wait activity for this latch can be minimized using the Redo Log Buffer tuning techniques discussed in Chapter 7.

Once you’ve determined that latch free waits are occurring in your instance, you can use the V$LATCH view, STATSPACK and REPORT.TXT output, and the OEM Performance Manager to identify whether any of the above latches are experiencing the waits on your system.

**Using V$LATCH to Identify Latch Contention**

You can query the V$LATCH dynamic performance view in order to identify whether any of the above latches have waits. Figure 9.1 shows a sample query on V$LATCH.

The output in Figure 9.1 shows the sources of the 59 latch-related wait events detected by our previous query on V$SYSTEM_EVENT. From a tuning perspective, we are only interested in latches listed in the previous section. Therefore, the session allocation, redo writing, and session timer latches can be ignored. The remaining latches’ wait events can be improved
by tuning the SGA memory structures associated with each latch as described in the previous section.

**FIGURE 9.1** Latch contention statistics in V$LATCH

```sql
SQL> SELECT name "WILLING TO WAIT LATCH", gets, misses, wait_time
2  FROM v$ latch
3  WHERE misses /= 0;
```

<table>
<thead>
<tr>
<th>WILLING TO WAIT LATCH</th>
<th>GETS</th>
<th>MISSES</th>
<th>WAIT TIME</th>
</tr>
</thead>
<tbody>
<tr>
<td>session allocation</td>
<td>5234099</td>
<td>1</td>
<td>4039</td>
</tr>
<tr>
<td>enqueue</td>
<td>1136681</td>
<td>2</td>
<td>3837</td>
</tr>
<tr>
<td>enqueue hash chains</td>
<td>990785</td>
<td>1</td>
<td>1690</td>
</tr>
<tr>
<td>redo writing</td>
<td>593464</td>
<td>1</td>
<td>28605</td>
</tr>
<tr>
<td>dml lock allocation</td>
<td>469073</td>
<td>1</td>
<td>1247</td>
</tr>
<tr>
<td>undo global data</td>
<td>1056063</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Library cache</td>
<td>4242732</td>
<td>11</td>
<td>1957</td>
</tr>
</tbody>
</table>

**Using STATSPACK Output to Identify Latch Contention**

The output from the STATSPACK utility includes two sources of information on latch wait activity. The first, a section entitled “Top 5 Wait Events” indicates whether latch free event waits have been occurring frequently in the instance. Figure 9.2 shows this section from a sample STATSPACK report.

**FIGURE 9.2** Latch free wait events in STATSPACK report

<table>
<thead>
<tr>
<th>Event</th>
<th>Waits</th>
<th>Unit Time (s)</th>
<th>% Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>control file sequential read</td>
<td>79</td>
<td>0</td>
<td>26.26</td>
</tr>
<tr>
<td>async disk 20</td>
<td>219</td>
<td>0</td>
<td>25.27</td>
</tr>
<tr>
<td>control file parallel write</td>
<td>214</td>
<td>0</td>
<td>23.58</td>
</tr>
<tr>
<td>latch free</td>
<td>56</td>
<td>0</td>
<td>20.84</td>
</tr>
<tr>
<td>log file parallel write</td>
<td>5</td>
<td>0</td>
<td>2.01</td>
</tr>
</tbody>
</table>

The output in Figure 9.2 indicates that waits for the latch free event occurred 56 times, accounting for 20.84 percent of the wait time. Since this statistic indicates that some latch contention is occurring in the instance, the next step is to determine specifically which latches are experiencing the waits. This information can also be found in a STATSPACK report as shown in Figure 9.3.
Chapter 9 • Tuning Contention

FIGURE 9.3  Latch wait statistics in STATSPACK report

The STATSPACK output shown in Figure 9.3 uses a “get miss ratio” as an indicator of how often a requested latch was not available when it was requested. The Pct Get Miss column indicates how often Willing-to-Wait latches were inaccessible when they were requested. The Pct NoWait Miss column indicates how often Immediate latches were not available when they were requested. The values for each of these ratios should be very low, ideally at 0.0. If the ratio is non-zero, waits for shared pool and library cache latches would indicate that the Shared Pool should be examined for possible performance enhancement; waits for cache buffers lru chain and cache buffers chains latches point to possible Buffer Cache performance enhancements; and redo allocation and redo copy latch waits are indicative of potential Redo Log Buffer tuning improvements.

Using REPORT.TXT to Identify Latch Contention

The REPORT.TXT file generated from the UTLBSTAT.SQL and UTLESTAT.SQL scripts also includes information on latch wait activity. Figure 9.4 shows sample latch activity output from a REPORT.TXT report.
The output in Figure 9.4 indicates that, unlike the “get miss ratio” found in STATSPACK output, REPORT.TXT uses a “hit ratio” to indicate latch performance. This ratio shows how frequently a requested latch was available when it was requested. The SGA component related to any latch whose value in the HIT_RATIO column is less than 1 would be a candidate for possible tuning.

Using OEM Performance Manager to Identify Latch Contention

The Performance Manager component of the Oracle Enterprise Manager Diagnostics Pack includes graphical representations of wait events, some of which can be related to latches. Figure 9.5 shows a sample Performance Manager screen that displays instance wait events.

The output in Figure 9.5 indicates that, during the Performance Manager monitoring period, there were no occurrences of waits related to the latches described in the previous section.
Tuning Latch Contention

DBAs do not tune latches. In fact, in Oracle9i, all `init.ora` parameters related to latch activity have been deprecated. Instead, DBAs use evidence of latch contention as an indicator of possible areas for tuning improvement in the database’s other structures, such as the SGA. Once identified, the structure related to the latch contention is then tuned, not the latch itself.

Free List Contention

In Chapter 8, “Tuning Disk I/O,” we discussed how Oracle segments use blocks to store their data. That chapter also discussed how each segment keeps a Free List that contains a listing of which of the segment’s blocks are able to accept new rows. You should recall that blocks on the Free List have not yet been filled to the PCTFREE value specified for the table. If your application has many users performing frequent inserts, the application user’s Server Process may experience waits when trying to access the Free
List for a frequently inserted table. These waits are called *Free List contention*. The tuning goal with regard to Free Lists is to minimize this type of contention by making sure that all processes can access a segment’s Free List without experiencing a wait.

**Oracle Objective**

Detect and resolve Free List contention

---

**Measuring Free List Contention**

Free List contention can be detected by querying `V$WAITSTAT`, `V$SYSTEM_EVENT`, and `V$SESSION_WAIT` dynamic performance views, and the `DBA_SEGMENTS` data dictionary view.

**Using V$SYSTEM_EVENT to Measure Free List Contention**

The `V$SYSTEM_EVENT` view shows statistics regarding wait events that have occurred in the database since instance startup. Occurrences of the *buffer busy wait* event indicate that Free List contention may exist in the database. The following query can be used to determine if *buffer busy waits* are occurring in your instance:

```sql
SQL> SELECT event, total_waits
2  FROM v$system_event
3  WHERE event = 'buffer busy waits';
```

<table>
<thead>
<tr>
<th>EVENT</th>
<th>TOTAL_WAITS</th>
</tr>
</thead>
<tbody>
<tr>
<td>buffer busy waits</td>
<td>1033</td>
</tr>
</tbody>
</table>

Finding the *buffer busy waits* in your `V$SYSTEM_EVENT` view does not in and of itself indicate a problem with Free List contention. The `V$WAITSTAT` view should also be examined to make sure that Free List contention is occurring during processing.

**Using V$WAITSTAT to Measure Free List Contention**

The `V$WAITSTAT` view contains statistics about contention for individual segment blocks. Table 9.2 describes the contents of the `V$WAITSTAT` dynamic performance view.
Tuning Contention

Non-zero wait statistics for two classes of blocks in particular, free list and segment header, indicate that Free List contention is occurring in the database. The sample following query on \texttt{V$WAITSTAT} returns the number of Free List waits that have occurred since instance startup:

\begin{verbatim}
SQL> SELECT class, count
    2  FROM v$waitstat
    3  WHERE class IN ('free list',
    4                  'segment header');
\end{verbatim}

\begin{table}[H]
\centering
\begin{tabular}{|l|l|}
\hline
Column Name & Description \tabularnewline
\hline
CLASS & Short description of the type of segment that the block belongs to \tabularnewline
COUNT & Number of times a wait to access a block occurred \tabularnewline
TIME & Total time spent waiting for access to blocks \tabularnewline
\hline
\end{tabular}
\caption{Contents of \texttt{V$WAITSTAT}}
\end{table}

The output from this query indicates that Free List related waits occurred 12 times for segment headers and four times for free lists.

Once you determine that Free List contention is present, you must next determine which segments are experiencing the contention. This can be achieved using a query on \texttt{V$SESSION\_WAIT} and \texttt{DBA\_SEGMENTS}.

Using \texttt{V$SESSION\_WAIT} and \texttt{DBA\_SEGMENTS} to Measure Free List Contention

The \texttt{V$SESSION\_WAIT} view, which contains statistics related to the waits experienced by individual sessions, can be joined to the \texttt{DBA\_SEGMENTS} view, which contains information about each segment in the database, to determine which segments are experiencing the Free List contention identified in the previous section. The following query shows how you can join these two views:

\begin{verbatim}
SQL> SELECT v.CLASS, v.COUNT
    2  FROM v$session_wait v
    3  JOIN dba_segments s
    4  ON v.NAME = s.NAME
    5  WHERE v.NAME IN ('free list',
    6                   'segment header');
\end{verbatim}

The output from this query indicates which segments are experiencing Free List contention.
tables using the P1, P2, HEADER_FILE, and HEADER_BLOCK columns:

```
SQL> SELECT s.segment_name, s.segment_type, s.freelists
  2  FROM dba_segments s, v$session_wait w
  3  WHERE w.p1 = s.header_file
  4  AND w.p2 = s.header_block
  5  AND w.event = 'buffer busy waits';
```

<table>
<thead>
<tr>
<th>SEGMENT_NAME</th>
<th>SEGMENT_TYPE</th>
<th>FREELISTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>EMPLOYEE</td>
<td>TABLE</td>
<td>1</td>
</tr>
</tbody>
</table>

The output from this query indicates that the EMPLOYEE table, which has only one Free List, is a source of Free List contention in the database.

**Tuning Free List Contention**

Once the segments experiencing Free List contention have been identified, you have two options for reducing contention. These options include: adding additional Free Lists to the segment or moving the segment to a tablespace that uses automatic segment-space management.

**Adding Additional Free Lists** A segment can have more than one Free List. However, by default, only one Free List is assigned to a segment at creation. If additional Free Lists need to be added to the segment in order to resolve Free List contention, then the segment can be altered:

```
ALTER TABLE hr.employee
STORAGE (FREELISTS 5);
```

Using the above syntax, the number of Free Lists assigned to the EMPLOYEE table would be increased to five.

**Moving Segments to Automatic Segment-space Managed Tablespaces**

One way to minimize Free List contention is to eliminate Free Lists from the process of determining which segment locks should be used to store a newly inserted row. This can be accomplished using a new Oracle9i feature called Automatic Segment-space Management. Tablespaces with the automatic segment-space management feature enabled utilize bitmaps in the tablespace’s datafile headers, instead of Free Lists, to manage the free block allocations for each segment in that tablespace. To use this feature to manage the free space allocations for the EMPLOYEE table, you must
first create a locally managed, automatic segment-space managed tablespace in which to store the EMPLOYEE table:

```
CREATE TABLESPACE appl_data
DATAFILE '/u01/oradata/PROD/appl_data01.dbf' SIZE 500M
EXTENT MANAGEMENT LOCAL
SEGMENT SPACE MANAGEMENT AUTO;
```

Then you need to move the EMPLOYEE table to the new tablespace using the MOVE command:

```
ALTER TABLE hr.employee MOVE TABLESPACE appl_data;
```

Any additional new segments created in an automatic segment-space managed tablespace will also use bitmaps instead of Free Lists for free block allocations.

Once moved to an Automatic Segment-space Managed tablespace, the segment’s value for the FREELISTS column in DBA_SEGMENTS view will be null.

## Lock Contention

As you have seen, Oracle uses latches to protect access to memory structures. Latches are held for only short periods of time, and waits for busy latches are also very brief.

Oracle’s locking mechanisms are similar to the latching mechanisms, but locks are used to protect access to data, not memory structures. Locks are also less restrictive than latches. In some cases, several users can share a lock on a segment. This is not possible with latches, which are never shared between processes.

Additionally, unlike latches, lock requests can be queued up in the order they are requested and then applied accordingly. This queuing mechanism is referred to as an enqueue. The enqueue keeps track of users who are waiting for locks, which lock mode they need, and in what order they asked for their locks.

Lock contention can occur any time two users try to lock the same resource at the same time. Lock contention usually arises when you have many users performing DML on a relatively small number of tables. When two users try modifying the same row in the same table, at the same time, lock contention results.

In general, locks are used to preserve data consistency. This means that the data a user is changing stays consistent within their session, even if other
users are also changing it. Oracle’s automatic locking processes lock data at the lowest possible level so as not to needlessly restrict access to application data. Since this type of locking model allows many users to access data at the same time, application users achieve a high degree of data concurrency.

Once taken out, a lock is maintained until the locking user issues either a COMMIT or a ROLLBACK command. Because of this, locks that are too long in duration or too restrictive in nature can adversely affect performance for other application users who need to access the locked data. In most cases, this locking is done at the row level. Using Oracle’s default locking mechanism, multiple users can do the following:

- Change different rows in the same table with no locking issues.
- Change the same row in the same table with enqueues determining who will have access to the row and when, and with System Change Numbers (SCN) deciding whose change will be the final one. In addition to the order in which locks were requested, the enqueue mechanism also keeps track of which users are waiting to obtain locks, and what type of lock they want to obtain.

By default, the maximum number of enqueues is derived from the value of the SESSIONS parameter in init.ora. This default is generally adequate, but the number of enqueues can also be tuned manually by setting the ENQUEUE_RESOURCES parameter in the init.ora.

Figure 9.6 demonstrates these two locking situations (two users updating different rows and the same row) when two users are updating and querying data in a table called DEPT.

As illustrated in this figure, when Joe issues his UPDATE statement, he locks the row associated with DEPT_ID 20. When Matt issues his UPDATE command, he locks the row associated with the Sales department. Since Joe and Matt are each locking a different row, neither is affected by the other’s lock. When Joe queries the DEPT table looking for the Sales department data, Matt’s lock will not prevent Joe from seeing the Sales data. However, since Matt has not yet committed his changes, Joe will use the before-image data in Matt’s rollback segment entry to build a read consistent view of the Sales data. Finally, when Matt tries to change the Accounting department’s name by updating the row whose DEPT_ID is 20, Matt will experience a wait, stuck behind Joe’s lock on the same row. Matt’s session will continue to wait until Joe issues either a COMMIT or a ROLLBACK.
The `init.ora` parameter `ROW_LOCKING` can be set to a value of `INTENT` if more restrictive, table-level locking is desired. The default value for this parameter is `ALWAYS`, which results in row-level locking.

Oracle uses two lock types to perform its locking operations: a DML or data lock and a DDL or dictionary lock. When performing these locking functions, Oracle also uses two different locking modes to achieve its goals of data concurrency and data consistency. The first lock mode, exclusive, is the most restrictive. An exclusive lock locks a resource until the transaction that took out the lock is complete. No other user can modify the resource while it is locked in exclusive mode. The second lock mode, share lock, is the least restrictive. A share lock locks the resource, but allows other users to also obtain additional share locks on the same resource. Each of these lock types and modes is discussed in the following sections.
DML or data locks are denoted by TM and are used to lock tables when users are performing INSERT, UPDATE, and DELETE commands. Data locks can either be at the table level or row level. Every user who performs DML on a table actually gets two locks on the table: a share lock at the table level and an exclusive lock at the row level. These two locks are **implicit**, meaning that Oracle performs the locking actions for the user. **Explicit locks** can also be taken out when performing DML commands. An explicit lock requires that you manually request a specified lock type. Table 9.3 compares modes used for DML locks.

**Table 9.3** Comparison of the Oracle DML Lock Modes

<table>
<thead>
<tr>
<th>Kind of Lock</th>
<th>Lock Mode (Lock Symbol)</th>
<th>Command Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Implicit</td>
<td>Row Exclusive (RX)</td>
<td>INSERT, UPDATE, DELETE Other users can still perform DML on any other row in the table.</td>
</tr>
<tr>
<td>Implicit</td>
<td>Table Row Share (RS)</td>
<td>SELECT . . . FOR UPDATE Other users can still perform DML on the rows in the table that were not returned by the SELECT statement.</td>
</tr>
<tr>
<td>Implicit</td>
<td>Share (S)</td>
<td>UPDATE and DELETE on parent tables with Foreign Key relationships to child tables Users can still perform DML on any other row in either the parent or child table as long as an index exists on the child table’s Foreign Key column.</td>
</tr>
<tr>
<td>Implicit</td>
<td>Share Row Exclusive (SRX)</td>
<td>DELETE on parent tables with Foreign Key relationships to child tables Users can still perform DML on any other row in either the parent or child table as long as an index exists on the child table’s Foreign Key column.*</td>
</tr>
<tr>
<td>Explicit</td>
<td>Exclusive (X)</td>
<td>LOCK TABLE . . . IN EXCLUSIVE MODE Other users can only query the table until the locking transaction is either committed or rolled back.</td>
</tr>
</tbody>
</table>

* In versions prior to 9i, Oracle will take out a more restrictive lock on the child table whenever appropriate indexes on the Foreign Key columns of a child table are not present.
By default, the number of DML locks is derived from the value of the TRANSACTIONS parameter in the init.ora. This default is generally adequate, but the number of DML locks can also be tuned manually by setting the DML_LOCKS parameter in the init.ora.

**DDL Locks**  
DDL or dictionary locks are denoted by TX and are used to lock tables when users are creating, altering, or dropping tables. This type of lock is always at the table level and is designed to prevent two users from modifying a table’s structure simultaneously. Table 9.4 shows a comparison of the different modes used for DDL locks.

**Table 9.4** Comparison of the Oracle DDL Lock Modes

<table>
<thead>
<tr>
<th>Lock Type</th>
<th>Lock Mode</th>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Implicit</td>
<td>Exclusive (X)</td>
<td>CREATE, DROP, ALTER</td>
<td>Prevents other users from issuing DML or SELECT statements against the referenced table until after the CREATE, DROP, or ALTER operation is complete</td>
</tr>
<tr>
<td>Implicit</td>
<td>Shared (S)</td>
<td>CREATE PROCEDURE, AUDIT</td>
<td>Prevents other users from altering or dropping the referenced table until after the CREATE PROCEDURE or AUDIT operation is complete</td>
</tr>
<tr>
<td>Implicit</td>
<td>Breakable Parse</td>
<td>-</td>
<td>Used by statements cached in the Shared Pool; never prevents any other types of DML, DDL, or SELECT by any user</td>
</tr>
</tbody>
</table>

**Real World Scenario**

**The Special Case of Deadlocks**

I was once working with an application user who was in the process of training a new employee to use the company’s Customer Relationship Management (CRM) system. While they were working together, I overheard the senior employee say, “Every once in a while I get this weird deadlock error. I don’t know what it is, but if you see that error, just enter
the data over again and it will work.” The error this user was describing is called a deadlock (or sometimes referred to as a “deadly embrace”). A deadlock error occurs whenever two or more users are waiting for locks on data that are already locked by the one of the two or more users. Sound confusing? A simple example will help clarify the definition:

1. Malka updates the EMP table by changing the last name of employee 20 to Jones:

   ```sql
   SQL> UPDATE emp
   2   SET last_name = 'Jones'
   3   WHERE id = 20;
   1 row updated.
   ```

2. Next, Jennifer updates the EMP table, changing the salary of employee 25 to $50,000:

   ```sql
   SQL> UPDATE emp
   2   SET salary = 50000
   3   WHERE id = 25;
   1 row updated.
   ```

3. Malka also updates employee 25, changing the last name to Smith. At this point, Malka is stuck waiting behind Jennifer’s lock on the same row, so Malka’s session hangs:

   ```sql
   SQL> UPDATE emp
   2   SET last_name = 'Smith'
   3   WHERE id = 25;
   ```

4. Now, Jennifer starts to update employee 20 (the row Malka locked in the first step) and tries to change their salary to $60,000:

   ```sql
   SQL> UPDATE emp
   2   SET salary = 60000
   3   WHERE id = 20;
   ```

This leaves Jennifer stuck waiting behind Malka’s lock on the same row. Both users are waiting for the other user to release their lock, but neither user can issue a COMMIT or ROLLBACK to release their locks because their sessions are hung up. This is a deadlock situation.
If there were no resolution to this situation, these two users would wait indefinitely for each other to release their locks. Luckily, Oracle automatically resolves deadlocks for you by simply rolling back one of the blocking statement, thus releasing one of the locks involved in the deadlock. The statement that gets rolled back will belong to the session that detects the deadlock situation. In our example, this would be Malka’s session, and she would see the following error:

ORA-00060: deadlock detected while waiting for resource

Malka’s session will also generate a trace file in the directory specified as the USER_DUMP_DEST in the init.ora. This trace file will indicate when, and on which table, the deadlock event occurred. A portion of Malka’s trace file is shown here:

*** 2002.02.13.10.07.14.777
ksqded1:  deadlock detected via did
DEADLOCK DETECTED
Current SQL statement for this session:
UPDATE emp
SET last_name = 'Smith'
WHERE id = 25

The following deadlock is not an ORACLE error. It is a deadlock due to user error in the design of an application or from issuing incorrect ad-hoc SQL. The following information may aid in determining the deadlock:
Deadlock graph:...

The advice offered by the senior employee to the new trainee was sound, because often times a subsequent attempt at entering the same data will not result in another deadlock error because the combination of lock events required to cause a deadlock are unlikely to occur consecutively. From the DBA’s perspective, you should monitor the USER_DUMP_DEST directory for trace files containing deadlock errors. If they are occurring frequently, you should consider revising the application code or modifying the business process flow to reduce the occurrences of the deadlocks.
Measuring Lock Contention

When left to its default mechanisms, Oracle generally does a very effective job of managing locking. When it does occur, lock contention can be identified using the V$LOCK and V$LOCKED_OBJECT dynamic performance views, the DBA_WAITERS and DBA_BLOCKS data dictionary views, and the OEM Performance Manager GUI. Each of these methods of monitoring lock activity is discussed below.

The DBA_BLOCKERS and DBA_WAITERS data dictionary views are created by executing the CATBLOCK.SQL script found in $ORACLE_HOME/rdbms/admin on Unix systems and %ORACLE_HOME%/rdbms/admin on Windows 2000 systems.

Using V$LOCK to Monitor Lock Contention

The V$LOCK dynamic performance view contains data regarding the locks that are being held in the database at the time a query is issued against the view. As shown below, this view can be joined to DBA_OBJECTS and V$SESSION to also see who is holding the lock and on which objects:

```sql
SQL> SELECT s.username,
          2          DECODE(l.type,'TM','TABLE LOCK',
          3                        'TX','ROW LOCK',
          4                        NULL) "LOCK LEVEL",
          5         o.owner, o.object_name, o.object_type
          6  FROM v$session s, v$lock l, dba_objects o
          7  WHERE s.sid = l.sid
          8  AND o.object_id = l.id1
          9  AND s.username IS NOT NULL;
```

Oracle Objective

Recognize Oracle errors arising from deadlocks

Oracle Objective

Use Oracle utilities to detect lock contention
Chapter 9 · Tuning Contention

This example shows that the users Scott and Joe are both locking the EMP table owned by Joe. If Joe had called to complain that his application session appeared to not be responding, this query might help you determine whether Joe’s transaction is stuck waiting for Scott to commit or roll back his transaction. However, this query does not explicitly indicate which user is blocking another user.

Using **V$LOCKED_OBJECT** to Monitor Lock Contention

The V$LOCKED_OBJECT view also lists all the locks currently held by every user on the system. However, it also includes blocking information showing which user is performing the locking transaction that is causing other application users to experience a wait. The following query can be used to determine who is locking which objects and the user who holds the blocking lock:

```sql
SQL> SELECT LPAD(' ',DECODE(l.xidusn,0,3,0))
2         ||l.oracle_username "User Name",
3         o.owner, o.object_name, o.object_type
4         FROM v$locked_object l, dba_objects o
5  WHERE l.object_id = o.object_id
6  ORDER by o.object_id, 1 desc;
```

<table>
<thead>
<tr>
<th>User Name</th>
<th>OWNER</th>
<th>OBJECT_NAM</th>
<th>OBJECT_TYPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCOTT</td>
<td>JOE</td>
<td>EMP</td>
<td>TABLE</td>
</tr>
<tr>
<td>BRENDA</td>
<td>JOE</td>
<td>EMP</td>
<td>TABLE</td>
</tr>
<tr>
<td>JOE</td>
<td>JOE</td>
<td>EMP</td>
<td>TABLE</td>
</tr>
<tr>
<td>ROBIN</td>
<td>APPS</td>
<td>PAYROLL</td>
<td>TABLE</td>
</tr>
<tr>
<td>REGI</td>
<td>APPS</td>
<td>PAYROLL</td>
<td>TABLE</td>
</tr>
</tbody>
</table>

The query output shows that Scott, Joe, and Brenda are all locking rows in Joe’s EMP table, while Robin and Regi are locking rows in the application PAYROLL table. Furthermore, the leftmost username indicates the user who holds the lock that is preventing the other users from continuing their work. In the first example, this means that Scott’s transaction on Joe’s EMP table is blocking
Brenda and Joe’s transactions. Likewise, Robin’s transaction on the PAYROLL

table is blocking Regi’s transaction on that same table. Techniques for resolving
these types of blocking transactions are covered in the next section.

**Using DBA_WAITERS to Monitor Lock Contention**

The DBA_WAITERS data dictionary view contains information about user ses-
sions that are waiting for locks to be released by other user sessions. Table 9.5
describes the contents of this view.

**TABLE 9.5** Contents of DBA_WAITERS

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WAITING_SESSION</td>
<td>The SID of the user session that is currently waiting to obtain a lock</td>
</tr>
<tr>
<td>HOLDING_SESSION</td>
<td>The SID of the user session that currently holds the lock that the user shown in WAITING_SESSION wishes to obtain</td>
</tr>
<tr>
<td>LOCK_TYPE</td>
<td>The type of lock held by the user associated with the HOLDING_SESSION</td>
</tr>
<tr>
<td>MODE_HELD</td>
<td>The lock mode held by the user associated with the HOLDING_SESSION</td>
</tr>
<tr>
<td>MODE_REQUESTED</td>
<td>The lock mode requested by the user associated with the WAITING_SESSION</td>
</tr>
<tr>
<td>LOCK_ID1</td>
<td>Internal lock identifier for the first (usually Share) lock being held by user associated with the HOLDING_SESSION</td>
</tr>
<tr>
<td>LOCK_ID2</td>
<td>Internal lock identifier for the second (usually Row Exclusive) lock being held by the user associated with the HOLDING_SESSION</td>
</tr>
</tbody>
</table>

Figure 9.7 shows how the DBA_WAITERS data dictionary view and the
V$SESSION dynamic performance view can be joined so that the username
associated with waiting and holding sessions can be displayed.
The output in Figure 9.7 shows that the user Matt is waiting to obtain an exclusive lock on the same table that user Joe currently has an exclusive transaction lock on.

**Using **DBA_BLOCKERS** to Monitor Lock Contention**

Often, the DBA is not concerned with what type of lock other users are trying to obtain; they are only concerned with determining which user is causing the lock contention so that the lock contention can be corrected using one of the techniques described in the next section. The **DBA_BLOCKERS** data dictionary view is the best source of this type of succinct locking information because the **DBA_BLOCKERS** view contains only one column, **HOLDING_SESSION**, which displays the SIDs of the user sessions that are blocking the lock requests of other application users. The following example shows how you can identify the application user that is blocking other users by joining **DBA_BLOCKERS** to **V$SESSION**:

```
SQL> SELECT s.username
2  FROM dba_blockers db, v$session s
3  WHERE db.holding_session = s.sid;
```

The output shows that the user Joe is currently blocking at least one another application user from obtaining a requested lock.

As an alternative to constructing your own queries on the **DBA_BLOCKERS** and **DBA_WAITERS** data dictionary views, you can also use the Oracle-supplied script, **UTLLOCKT.SQL** to display locking information. This script is found in `$ORACLE_HOME/rdbms/admin` on Unix systems and `%ORACLE_HOME%\rdbms\admin` on Windows 2000 systems.
Using OEM Performance Manager to Monitor Lock Contention

The Performance Manager component of the Oracle Enterprise Manager Diagnostics Pack includes several graphical representations of lock activity. Figure 9.8 shows a sample of Performance Manager’s Lock Manager screen.

The output in Figure 9.8 shows that the application user HR holds several locks on the JOB_HISTORY table.

The next section discusses how to resolve lock contention problems once you have identified them using the above techniques.

Tuning Lock Contention

As mentioned earlier, lock contention occurs when two users try to lock the same resource in exclusive mode at the same time. When using Oracle’s default locking mechanisms, lock contention is usually only a problem when your application has many users performing DML on a relatively small
number of tables. However, since Oracle’s enqueue mechanism assigns locks in the order in which they were requested, a user who wishes to lock a row that is already locked by another user may experience a delay for an indeterminate length of time while waiting for the other user to commit or roll back their transaction. Since this delay will generally be perceived as poor database performance by the user, you will need to resolve the contention using one of these methods:

- Change the application code so less-restrictive locking is used
- Contact the blocking user and ask them to commit or roll back their changes
- Resolve the lock contention using an SQL command

Oracle Objectives

<table>
<thead>
<tr>
<th>Describe possible causes of contention</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prevent locking problems</td>
</tr>
</tbody>
</table>

Changing Application Code to Resolve Lock Contention

Lock contention problems related to application SQL code arise from two situations: application developers have coded too many long transactions, and/or application developers have explicitly coded restrictive lock levels.

**Coding Long Transactions** If application developers code long-running transactions that do not commit regularly, and many concurrent users are using the application, then the likelihood of experiencing lock contention is greatly increased. To reduce the occurrences of lock contention, application design should include logical commit points so that transactions do not hold unnecessarily long locks.

**Restrictive Explicit Locks** If application developers have explicitly coded locks that are more restrictive than the Oracle defaults, then the likelihood of experiencing lock contention is greatly increased. Specifically, lock contention can occur when developers code frequent `LOCK TABLE IN EXCLUSIVE MODE` commands within the application. Unless there is a specific application need for it, developers should let Oracle’s default locking mechanisms handle all locks in order to minimize lock contention.
Many commercial business applications are designed so that the customer can use the products with the database software of their choosing. Therefore, these products are not designed to take advantage of the unique features of any particular database vendor. Since not all relational databases share Oracle’s propensity for row-level locking, developers who write these commercial packages sometimes explicitly code locks into the application that are higher than Oracle’s default locking mechanisms in an effort to make the application more portable across databases. If these explicit locks cause locking contention to occur, there is very little a DBA can do to correct it, because the application code may either be unavailable or too costly to modify.

**Contacting Users to Resolve Lock Contention**

User sessions that block other user’s requested locks usually arise when an application user is interrupted while performing their work. Phone calls, meetings, and meal breaks can all cause an application user to suspend their activities for long periods, leading to lock contention. Once a blocking user session has been identified, you can try to contact the offending application user via e-mail or telephone in order to get the user to either commit or rollback their work. Doing so will release the locks and resolve the lock contention.

Another way you can prevent this problem is to disconnect users who are idle for a specified period of time through the use of Oracle’s Profile feature. By creating a profile that is allowed a specified period of inactivity before disconnecting, and then assigning each user to this profile, you can ensure that users who start transactions will have those transactions rolled back if they are idle for too long.

See Chapter 24, “Managing Users and Resources,” of *Oracle9i Database Administrator’s Guide Release 1 (9.0.1)* (Part Number A90117-01) for details on how to use Oracle’s Profile feature.

**Using SQL Commands to Resolve Lock Contention**

If a blocking user cannot be contacted using any of the means listed in the previous section, then you can use the `ALTER SYSTEM KILL SESSION` command
to kill the blocking session. This command disconnects the blocking user’s session and rolls back any uncommitted transactions they had open. Before you can kill a blocking user’s session, you must identify the SID and SERIAL# for their session using V$SESSION. The following example shows how to use DBA_BLOCKERS and V$SESSION to determine the SID and SERIAL# of a blocking user:

```
SQL> SELECT s.username, s.sid, s.serial#
       2 FROM dba_blockers db, v$session s
       3 WHERE db.holding_session = s.sid;
```

<table>
<thead>
<tr>
<th>USERNAME</th>
<th>SID</th>
<th>SERIAL#</th>
</tr>
</thead>
<tbody>
<tr>
<td>JOE</td>
<td>12</td>
<td>8881</td>
</tr>
</tbody>
</table>

If all users connect to the application using the same Oracle username, try adding the V$SESSION columns OSUSER, MACHINE, or PROGRAM to your query to help identify which user’s session you should kill.

---

**Oracle Objective**

Resolve contention in an emergency

Once you have identified the SID and SERIAL# for the session to be killed, you can issue the ALTER SYSTEM command to actually kill the session:

```
SQL> ALTER SYSTEM KILL SESSION '12,8881';
```

The user whose session was killed does not get an immediate indication that this has occurred unless the user is accessing the database at that moment. If they are idle when their session is killed, they will not receive the ORA-00028 “Your Session Has Been Killed” error message until they initiate new activity against the database.

Another technique to reduce locking issues is to use the Oracle9i SQL command ALTER SYSTEM QUIESCE RESTRICTED. If Resource Manager is enabled, issuing this command will cause all inactive user sessions to release their database resources, including locks.
Summary

This chapter discussed the tuning-related aspects of contention for latches, Free Lists, and locking. Latches are used to protect access to memory structures. Although there are many types of latches, the Shared Pool, Library cache, Cache Buffers LRU Chain, Cache Buffers Chains, Redo Allocation, and Redo Copy latches are the most useful in terms of contention tuning. Latch contention can be identified by examining V$SYSTEM_EVENT, V$LATCH, STATSPACK and REPORT.TXT output, and the OEM Performance Manager. Latches are not tuned directly. Instead, the components of the SGA that are experiencing latch contention are tuned using the techniques described in previous chapters.

Free List contention can also be a performance tuning consideration on systems that perform frequent insert activity. The number of Free Lists for a table can be changed by dropping and re-creating the table or by using the ALTER TABLE command. The dynamic performance views V$SYSTEM_EVENT, V$SESSION_WAIT, and V$WAITSTAT can be used to determine if Free List contention is occurring in your database.

Locking issues can also be a source of performance tuning problems. While Oracle’s default locking mechanisms are primarily at the row level, performance issues can still occur when applications code excessively high lock levels, long transactions, or when users leave transactions uncommitted for extended periods. The Oracle Server automatically resolves deadlocks whenever they occur. You can use the V$LOCK and V$LOCKED_OBJECT dynamic performance views and the DBA_WAITERS and DBA_BLOCKERS data dictionary views to monitor lock contention in the database. You can resolve excessive lock contention by modifying application code, notifying blocking users, and killing the session of blocking users.

Exam Essentials

Be able to describe the purpose of latches and how to monitor their performance. Know which system events are related to each type of latch. Understand what changes should be made to each SGA component if latch contention is present.

Know how to detect Free List contention. Be able to describe when Free List contention is most likely to occur and how it can be resolved once detected.
Understand Oracle’s lock types and locking modes. Know which lock types are the least restrictive and the most restrictive. Be able to describe how to monitor Oracle’s locking mechanisms and how to resolve lock contention.

Key Terms

Before you take the exam, make sure you are familiar with the following terms:

- Automatic Segment-space Management
- contenton
- data concurrency
- data consistency
- data lock
- deadlock
- dictionary lock
- enqueue
- Free List contenton
- Immediate
- latch
- lock contenton
- Willing-to-Wait
Review Questions

1. Which of the following latch types allows a user to continue carrying out other processing if the latch requested was not immediately available?
   A. Immediate
   B. Instant
   C. Willing-to-Wait
   D. Unwilling-to-Wait

2. Which of the following latch types waits before attempting to re-obtain a busy latch?
   A. Immediate
   B. Instant
   C. Willing-to-Wait
   D. Unwilling-to-Wait

3. Which of the following mechanisms is used to protect access to memory structures?
   A. Enqueue
   B. Lock
   C. Latch
   D. init.ora

4. The number of times a user Server Process has successfully obtained a Willing-to-Wait latch without waiting is recorded in which column in V$LATCH?
   A. MISSES
   B. GETS
   C. SLEEPS
   D. IMMEDIATE_GETS
5. The number of times a user Server Process has successfully obtained an Immediate latch without waiting is recorded in which column in V$LATCH?
   A. MISSES
   B. GETS
   C. SLEEPS
   D. IMMEDIATEGETS

6. Which latch is used by Server Processes when looking for room to copy blocks in the Database Buffer Cache?
   A. LRU latch
   B. Buffer Cache latch
   C. Cache Buffers LRU Chain latch
   D. Chain LRU latch

7. Which of the following statements would not obtain a row lock?
   A. SELECT FOR UPDATE
   B. INSERT
   C. DELETE
   D. DROP TABLE

8. What is the desired hit ratio for the Database Buffer Cache LRU List?
   A. 99 percent
   B. 85 percent
   C. 1 percent
   D. 5 percent

9. Segment blocks go on and off the Free List according to what values associated with the table?
A. INITIAL/NEXT
B. PCTFREE/PCTUSED
C. TABLESPACE/DATAFILE
D. MINEXTENTS/MAXEXTENTS

10. Free List contention can be detected using which of the following dynamic performance views? (Choose all that apply.)
A. V$SYSTEM_EVENT
B. V$SESSION
C. V$SESSION_WAIT
D. V$WAITSTAT

11. On multiple CPU servers, a process requesting an in-use latch will do what on the CPU before releasing the CPU and trying again?
A. Sleep
B. Move on to other processing
C. Spin
D. Timeout

12. What term refers to the process of making sure that a user sees a consistent view of the database throughout their transaction?
A. Data concurrency
B. Data constancy
C. Data consistency
D. Data contention

13. By default, most Oracle exclusive locks related to DML are at what level?
A. Table level
B. Column level
C. Block level
D. Row level
14. When the SELECT FOR UPDATE command is used, what type of lock is taken out of the selected rows?
   A. Table lock
   B. Row exclusive
   C. Table row share
   D. Exclusive

15. Which of the following lock types is the most restrictive?
   A. Exclusive
   B. Unique
   C. Exclusion
   D. None of the above is restrictive

16. Which lock type is taken out when creating, dropping, or altering a table?
   A. Restrictive
   B. Exclusive
   C. Explicit
   D. Shared

17. Review the following code segment:

   SQL> SELECT s.username
        2  FROM dba_blockers db, v$session s
        3  WHERE db.holding_session = s.sid;

<table>
<thead>
<tr>
<th>USERNAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>JOE</td>
</tr>
</tbody>
</table>

   What statement below best describes the information in the code segment?
A. Joe is a blocker.
B. Joe is a waiter.
C. Joe cannot obtain a lock.
D. There is not enough information provided to determine anything meaningful.

18. If your application has many users performing frequent inserts into a table, what type of contention may become a problem?
   A. Redo Allocation Latch contention
   B. Free List contention
   C. Redo Copy Latch contention
   D. Library Cache Latch contention

19. If you wanted to kill the session for a user Jim, and the contents of V$SESSION showed his SID and SERIAL# to be 20 and 2369, what command would you issue to kill the session?
   A. ALTER SYSTEM KILL SESSION (20,2369);
   B. ALTER SESSION KILL SYSTEM '20,2369';
   C. ALTER SYSTEM KILL SESSION '20', '2369';
   D. ALTER SYSTEM KILL SESSION '20,2369';

20. When an Oracle deadlock occurs, where is the deadlock information recorded after it is resolved?
   A. Alert log
   B. V$DEADLOCK
   C. User trace file
   D. PMON trace file
Answers to Review Questions

1. A. Immediate latches continue processing while waiting for the latch to become available. Willing-to-Wait latches will sleep between attempts at acquiring the latch.

2. C. Willing-to-Wait latches sleep before trying to obtain a previously requested latch that was busy.

3. C. Latches are used to protect access to memory structures like the LRU lists and the Redo Log Buffer.

4. B. The GETS column of V$LATCH shows the number of times that a Willing-to-Wait latch was obtained without waiting.

5. D. The IMMEDIATE_GETS column of V$LATCH shows how often Immediate latches were obtained without waiting.

6. C. User Server Processes search the LRU list using the Cache Buffers LRU Chain latch when placing copies of blocks into the Database Buffer Cache.

7. D. The DROP TABLE statement would cause a DDL Lock to be obtained on the table to ensure that no session can access the table as it is being dropped. All the other statements would obtain some type of row level lock.

8. A. A hit ratio of at least 99 percent is desired for the Database Buffer Cache LRU List.

9. B. Segment blocks are taken off the Free List when they are filled to PCTFREE and placed back on the Free List when they are emptied to PCTUSED.

10. A, C, D. The V$SYSTEM_EVENT view contains statistics on the buffer busy waits event. V$WAITSTAT contains statistics about waits for Free List blocks. V$SESSION_WAIT can be joined to DBA_SEGMENTS to determine which segments are experiencing Free List contention.

11. C. The process will spin on the CPU a number of times until releasing the CPU. The spin process is a type of internal wait mechanism.

12. C. Oracle uses rollback segments and SCNs to ensure that users always see a read consistent view of the database.
13. D. While Oracle does take out a share lock at the table level, an exclusive lock on the data being changed is only taken out at the row level.

14. C. The SELECT FOR UPDATE command takes out a table row share lock, preventing other users from performing DML on the selected rows.

15. A. An exclusive lock is the most restrictive type, allowing other users to perform only SELECT statements until the lock is released.

16. B. Performing DDL on a table requires that an exclusive lock be taken out on that table.

17. A. By querying DBA_BLOCKERS joined to V$SESSION, we can tell that Joe has obtained and is holding a lock on an object that another session is requesting. Joe is the blocker preventing the other session from obtaining the lock.

18. B. Free List contention can become a problem because many resources are trying to obtain information from the segment header as to which blocks are available to store newly inserted rows. The Free List in the segment header contains this information.

19. D. The values for SID and SERIAL# must be enclosed in single quotes and separated by commas when killing a session with the ALTER SYSTEM command.

20. C. The user whose session detects the deadlock and has their transaction rolled back will generate a trace file in the USER_DUMP_DEST location specified in init.ora.
Operating System Tuning

ORACLE9i PERFORMANCE TUNING EXAM
OBJECTIVES COVERED IN THIS CHAPTER:

✓ Describe different system architectures
✓ Describe the primary steps of OS tuning
✓ Identify similarities between OS and DB tuning
✓ Understand virtual memory and paging
✓ Explain the difference between a process and a thread
✓ Set up Database Resource Manager
✓ Assign users to Resources Manager groups
✓ Create resource plans within groups

Note: Exam objectives are subject to change at any time without prior notice and at Oracle's sole discretion. Please visit Oracle’s Certification website (http://www.oracle.com/education/certification/) for the current exam objectives listing.
he previous chapters have explained how to identify potential performance bottlenecks in the logical and physical components of the Oracle architecture. You have also seen what steps can be taken to enhance performance of each of these components. However, every Oracle database resides within the confines of the host server’s operating system and hardware resources. Therefore, many non-Oracle, server-related factors should also be considered when taking a holistic approach to database performance tuning.

This chapter addresses the impact that these non-Oracle factors have on database performance, and how to enhance performance by utilizing Oracle’s resource management features to manage these server resources.

Understanding Server Resources

There are three primary server resources that impact the performance of Oracle databases:

1. Memory
2. Disk I/O
3. CPU

In general, this is the order in which these OS resources should be tuned. Since tuning disk I/O was discussed in detail in Chapter 9, “Tuning Contention,” this chapter will concentrate on tuning considerations related to the first and last areas: Memory and CPU.
Understanding and Tuning Server Memory

When an instance is started, the SGA and all of its background processes are created in the server’s main memory. This memory area is used not only by the SGA and background processes, but also by the operating system and any other processes that may be running on the server. Examples of some of these non-Oracle processes that consume server memory are shown below.

Unix or Windows 2000 Kernel Code Every operating system loads its essential operating executables into memory when the server is booted. These executables, referred to as the OS kernel, are used to manage the operating system’s essential memory and I/O functions. On Unix systems, the kernel’s components are tunable by the system administrator in much the same way that Oracle’s init.ora parameters are tunable by the DBA. Examples of three important Oracle-related Unix kernel parameters are shown in Table 10.1.

<table>
<thead>
<tr>
<th>TABLE 10.1 Sample Unix Kernel Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Parameter</strong></td>
</tr>
<tr>
<td>SHMMAX</td>
</tr>
<tr>
<td>SHMMIN</td>
</tr>
<tr>
<td>MAXFILES</td>
</tr>
<tr>
<td>NPROC</td>
</tr>
</tbody>
</table>
Consult your OS-specific Oracle documentation for specific recommendations regarding the appropriate kernel parameter changes for your server’s operating system.

Kernel parameter modifications and kernel rebuilds should be attempted only by experienced Unix system administrators. Improper kernel parameter adjustments can lead to an unstable or unusable OS.

**Unix and Windows 2000 Services**  Every operating system has OS-level services that are started when the server is booted. Examples of these services on Unix systems include print services; logical volume managers; X-Windows services; and telnet, cron, and ftp daemons. Examples of Windows 2000 OS services include the Event Log, Plug and Play, and Print Spooler services. Each of these services consumes some of the server’s available memory.

**Application or Web Servers**  Some database servers also hold the application and web servers that provide the database application’s business logic or presentation layer. For example, one configuration of Oracle’s own E-Business Applications Suite places the application server, forms server, reports server, and database server all on the same physical machine. Each of these processes also consumes server memory.

**Software Agents**  Similar to Oracle’s own Intelligent Agent and OEM Console, many enterprise-wide network monitoring and backup systems utilize Simple Network Management Protocol (SNMP) agents to communicate from the server being managed back to the management or monitoring software. These agents will consume some of the server’s available memory while running.

**Application Programs**  Some servers will occasionally have application software packages installed on them. These applications can range from web browsers and PERL applications on Unix boxes to full-blown office suites on Windows 2000 servers. In general, it is bad practice to use your database server to run application software since this too will negatively impact the memory resources available to Oracle.

Every server has two types of memory available to store the OS kernel components and any Oracle memory structures and background processes.
The first of these two types of memory is physical memory. *Physical memory* is represented by the amount of memory in the server’s physical memory chips. Modern servers have large amounts of physical memory—2GB and 4GB of RAM are common. This available memory is divided up into smaller pieces called pages. Much like Oracle extents stored within tablespaces, the allocation of processes and data to these pages is managed by the operating system via a mechanism called a page table. If the physical memory pages are full, a server’s OS may have to temporarily remove pages or processes from memory in order to make room for other requested pages or processes. This process is referred to as paging or swapping. Table 10.2 compares paging and swapping.

<table>
<thead>
<tr>
<th>TABLE 10.2 Paging versus Swapping</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Paging</strong></td>
</tr>
<tr>
<td>An individual memory page is temporarily removed from memory and written to disk in order to make room for another requested page.</td>
</tr>
</tbody>
</table>

When paging or swapping occurs, the second type of memory, *virtual memory*, is used to store the on-disk copy of the page or process that was removed from physical memory. Much like Oracle SGA tuning, the primary goal of OS tuning is to maximize the frequency with which requested memory pages are already cached in memory when requested so that they do not need to be read from disk (i.e., virtual memory).

The techniques used to monitor paging and swapping vary by operation system. Table 10.3 shows examples of Unix commands that can be used to monitor paging and swapping. See your Unix man pages for details on the output returned from each of these commands.

<table>
<thead>
<tr>
<th>TABLE 10.3 Commands to Monitor Paging and Swapping on Unix Systems</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Command</strong></td>
</tr>
<tr>
<td>ipcs</td>
</tr>
<tr>
<td>ps</td>
</tr>
</tbody>
</table>
Windows 2000 systems utilize paging only for their memory management; swapping is not considered. The Kernel Memory section of the Performance tab in the Windows 2000 Task Manager utility is an easy way to monitor paging activities on a Windows 2000 server. Figure 10.1 shows a sample Windows 2000 Task Manager screen.

**Figure 10.1** Windows 2000 Task Manager

The output in Figure 10.1 indicates that total physical memory in the server is 1,048,032KB (1GB), with 913,480KB (892MB) used and...
267,056KB (250MB) of memory free. The output also shows that 40,624KB (40MB) of memory has been paged to disk during processing.

Because reads from disk (virtual memory) are many times slower than reads from memory (physical memory), excessive OS paging or swapping can cause poor database performance as well.

There are several techniques available for tuning the server memory in concert with Oracle’s SGA. These techniques include locking the SGA and using Intimate Shared Memory.

**SGA Locking** Setting the `init.ora` parameter `LOCK_SGA=TRUE` will prevent the OS from paging or swapping any portion of the SGA by locking the entire SGA in physical memory. The default value for this parameter is FALSE.

The `LOCK_SGA` parameter is not available on Windows 2000 servers.

**Intimate Shared Memory** Database servers running the Sun Solaris operating system can also make use of Intimate Shared Memory (ISM) to help tune OS paging and swapping activities. Enabling ISM allows multiple OS processes to share access to the OS page tables that contain entries for shared memory areas and, at the same time, locks the SGA into physical memory.

**Understanding and Tuning Server CPUs**

The server’s CPU ultimately performs all the processing that occurs on a server. If a server has multiple CPUs, then the processing tasks are shared among the available CPUs as assigned by the server’s operating system. The number of available CPUs is often determined by the architecture that the server itself is based on. Table 10.4 describes five common server architectures.
Oracle is also a very CPU-aware product. If new CPUs are added to a database server, Oracle will dynamically change the default settings for several CPU-related parameters. Examples of some of these parameters are shown in Table 10.5.

**Table 10.4** Common Server Architectures

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Uniprocessor</td>
<td>Servers with a single CPU and memory area. I/O, bus, and disk resources are used exclusively by this server.</td>
</tr>
<tr>
<td>Symmetric Multiprocessor (SMP)</td>
<td>Servers with two or more CPUs, but one memory area. I/O, bus, and disk resources are used exclusively by this server.</td>
</tr>
<tr>
<td>Massively Parallel Processor (MPP)</td>
<td>Two or more servers connected together. Each server has its own CPU, memory area, I/O bus, disk drives, and a copy of the OS.</td>
</tr>
<tr>
<td>Non-Uniform Memory Access (NUMA)</td>
<td>Two or more SMP systems connected together. Each server has its own CPU, but the memory areas for each server is shared as one large memory area across all servers. A single copy of the OS is run across all servers.</td>
</tr>
</tbody>
</table>

Oracle is also a very CPU-aware product. If new CPUs are added to a database server, Oracle will dynamically change the default settings for several CPU-related parameters. Examples of some of these parameters are shown in Table 10.5.

**Table 10.5** CPU-related `init.ora` Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU_COUNT</td>
<td>Specifies the number of CPUs in the server.</td>
</tr>
<tr>
<td>PARALLEL_THREADS_PER_CPU</td>
<td>Used to establish the default degree of parallelism when using Oracle’s parallel execution features.</td>
</tr>
<tr>
<td>FAST_START_PARALLEL_ROLLBACK</td>
<td>The number of parallel rollback processes is derived from this value (HIGH, LOW, FALSE) and the number of CPUs.</td>
</tr>
</tbody>
</table>
While any database server will usually benefit from the presence of additional CPUs, the performance improvement will be most dramatic where Oracle’s parallel execution features are being employed. Some examples of Oracle’s parallel execution options include: Parallel Query, Parallel DML, Parallel ANALYZE, and Parallel Index Creation.

**Parallel Query** When configured, Oracle’s Parallel Query feature allows a single query to spawn multiple query slaves. Each of these query slaves works on a portion of the query before the individual results are combined into a final result set. The number of query slaves that can be spawned is closely related to the number of server CPUs. Parallel Query’s CPU usage can be controlled through the use of the Resource Manager feature discussed later in this chapter.

**Parallel DML** Some insert, update, and delete activities can be performed in parallel using Oracle’s Parallel DML (PDML) features. Examples of PDML include CREATE TABLE AS SELECT statements and use of the /*+ PARALLEL */ SQL hint.

**Parallel ANALYZE** Table and index statistics can be gathered in parallel if the DBMS_STATS package is being used.

**Parallel Index Creation** Indexes that are being created on large tables can be created in parallel if the PARALLEL keyword is included in the CREATE_INDEX command. Including the PARALLEL keyword allows Oracle to start multiple processes that work together to create the index more quickly than if the index were built by a single process.

CPU tuning should begin with monitoring CPU utilization rates. The Unix `top` command can be used to monitor CPU utilization. Figure 10.2 shows sample output from an HP-UX `top` command.
The top portion of the output in Figure 10.2 shows that this server has two CPUs (0 and 1) that are currently 99.8 percent and 98.6 percent idle, respectively. Further down in the output, you can see that six Oracle processes are running, each utilizing one or the other of the CPUs to perform their processing.

Oracle recommends keeping CPU utilization below 90 percent for optimal performance. Oracle also recommends configuring multiple CPU servers so that each CPU performs approximately the same amount of processing.

The Windows 2000 Task Manager’s Performance tab discussed in the previous section can also be used to monitor CPU utilization, as shown in Figure 10.3.
The output shown in Figure 10.3 shows that the server contains one CPU—as evidenced by the single line on the CPU Usage History graph. This CPU is currently 83 percent utilized. The Processes tab of Task Manager can be used to identify how many CPU resources each individual process is consuming. Figure 10.4 shows sample output of a Task Manager Processes tab.

The output in Figure 10.4 shows that the top two processes in terms of CPU usage are the Oracle kernel (oracle.exe) and SQL*Plus (sqlplus.exe).

Oracle recommends that the majority of CPU activity be attributed to application processing—not OS processing. High CPU usage by the OS can be an indicator of poor server memory management.
Understanding and Tuning Processes and Threads

Closely related to CPU utilization is the concept of Oracle process and thread management. Unix and Windows 2000 operating systems behave differently in terms of how these processes are created.

On Unix systems, each Oracle-related process (e.g., PMON, DBW0, sqlplus) runs on the OS as a separate and distinct process. Each process consumes its own memory and CPU resources.

On Windows 2000 systems, each Oracle-related process runs as a separate thread within a single OS process called oracle.exe. A thread is an independent sequence of instructions that executes within a single OS process. The process can contain multiple threads.
One advantage of threads is the speed of the communication between the threads within the process. This communication is faster than the comparable communication between independent OS processes.

Understanding Resource Manager

As you have seen in the preceding section, Oracle utilizes many hardware resources while application users are accessing an instance. In cases where these resources are scarce, the DBA must allocate the resources so that the needs of the users are met most effectively. In Oracle versions prior to Oracle8i, DBAs could only manage user resource consumption through the use of profiles. However, profiles did not offer enough granularity to be considered a complete resource management solution.

Oracle8i introduced a new feature, Resource Manager, which was designed to improve the allocation and management of server resources needed by application users. Utilizing Resource Manager, a DBA can now create resource groups and assign them specific resource limits. These resource groups can in turn be assigned to application users who inherit the resource limits assigned to that group.

A new set of data dictionary tables is also available to monitor the resources and users assigned to each group. In Oracle9i, it is possible to control numerous aspects of application processing via Resource Manager. Several are related to CPU consumption. For example, Resource Manager allows the DBA to limit the amount of CPU that an individual application user can consume and the number of slave processes that an application user can spawn when performing Parallel Query operations. However, Resource Manager can also be used to control resource allocations such as a user’s maximum amount of undo segment usage and number of sessions against the database.

Using Resource Manager to control these resources allows the DBA to ensure that members of higher-priority resource groups will obtain sufficient CPU, memory, undo segment, and session resources even when many users are accessing the system. Conversely, other user groups can have their access to these database resources restricted when other, preferred resource groups are working. For example, users performing business-critical operations (like order processing) can be guaranteed sufficient access to CPU resources.
even if other users are performing complex queries for business intelligence reporting. Resource Manager also allows the DBA to dynamically change the allocation of these resources by changing the instance’s active resource plan. This gives the DBA the ability to have one resource plan active during regular, business-hours processing and a second resource plan during off-hours batch processing.

Oracle Resource Manager is only available in the Oracle9i Enterprise Edition.

The Oracle9i Resource Manager uses three components to achieve the goal of effective resource management: resource consumer groups, resource plans, and resource plan directives. Table 10.6 describes each of these Resource Manager components.

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resource consumer group</td>
<td>A group of users who have similar needs for database resources and who are governed by similar rules for allocating resources to those users. A user can be a member of several resource groups, but only one group can be active at a time.</td>
</tr>
<tr>
<td>Resource plan</td>
<td>A collection of resource plan directives that are used to determine how resources are allocated to users.</td>
</tr>
<tr>
<td>Resource plan directive</td>
<td>Specifies how many resources are allocated to each resource plan.</td>
</tr>
</tbody>
</table>

**Resource Manager Resources**

Resources that can be controlled through the use of Resource Manager include these:

1. The amount of CPU allocated to each user who is a member of the resource consumer group
2. The degree of parallelism allowed for Parallel Queries performed by a user who is a member of the resource consumer group
3. The amount of undo segment space that a user who is a member of a resource group is allowed to consume when performing a long running transaction

4. The total number of active, concurrent sessions that a given resource consumer group is allowed to have at one time

5. The maximum expected total time of a database action taken by a user who is a member of the resource consumer group

The manner in which each of the above resources is allocated is detailed in the following sections.

Managing CPU Resources

The CPU resources are allocated using the Emphasis method. The resources associated with Parallel Query are assigned using the Absolute method. Both methods use a relative scale, consisting of values from 1 to 8 to determine what priority should be assigned to a particular user’s request for resources. Using this relative scale, 1 represents the highest priority, 8 the lowest. The priority granted to a user is based on the resource group (within a resource plan) that the user is a member of. Each of these methods, and the techniques for employing them for resource management, will be discussed in the following sections.

How the Emphasis Method Manages CPU Resources

The Emphasis method uses percentages, expressed in terms of a value between 0 and 100, to assign CPU resources to each of the eight priority levels. The percentage assigned to a resource consumer group at a given level determines the maximum amount of CPU that users in that consumer group can consume at that level. If excess CPU resources remain after all users at that level have been serviced, then the remaining CPU resources will be made available to users at the next level, and so on. An analogy may help illustrate this concept.

You can think of the Resource Manager mechanism for managing CPU resources like an eight-story hotel whose rooms are full of hungry guests who have all ordered dinner via room service. It is the job of the hotel concierge (Resource Manager) to decide how to allocate the hotel’s ten bellhop resources (CPUs) to deliver the meals to the guests’ rooms. Because the concierge wants to try to meet the needs of all the guests, he may decide to allocate the bellhop resources in this way.

The guests on the top floor are given the highest priority because they are paying the highest rates for their penthouse accommodations. Therefore the
The concierge decides to allow these guests to consume up to 50 percent of the available resources (five bellhops) to service their requests for room service on that floor.

The guests on the seventh floor are given the second highest priority. The concierge decides to allow these guests to consume up to 30 percent of the available resources (three bellhops) to service their requests for room service on that floor.

The guests on the remaining floors are granted equal priority; each floor is allowed to use up to 20 percent of the available resources (two bellhops) to deliver the requested meals on those floors.

If the guests on the top floor are already being adequately served by only three bellhops, then the remaining two bellhops that had been allocated to servicing the top floor will be made available to guests on the seventh floor. If the guests on the seventh floor do not need the bellhops, they will be offered to the guests on the remaining floors. However, if the top floor guests should suddenly need the extra bellhops, any bellhops on the lower floors will be reassigned to guests on the top floor as soon as they complete any delivery in progress.

The resources within resource groups do not use the Emphasis method to allocate CPU resources. Instead, the CPU resources are allocated in a "round-robin" fashion. In other words, CPU resources are first offered to the first consumer group, then the next, then the next and so on, until they are offered to the first consumer group again.

This previous analogy is conceptually identical to the way Resource Manager controls requests for access to the server’s CPU resources. For example, suppose we wanted to create a resource consumer group called POWER_USERS for your most sophisticated application users. Within this group, you want to control access to CPU resources by dividing the POWER_USERS resource consumer group into two resource plans: FUNCTIONAL and TECHNICAL. These two resource plans can be assigned CPU priority on up to eight levels of granularity. Figure 10.5 conceptually shows how this resource allocation can be broken down.

Notice that the TECHNICAL resource plan is allocated 100 percent of CPU at the first level, 50 percent at the second and third levels, 25 percent at the fourth level, and zero percent at the remaining levels. This means that users in the TECHNICAL resource plan will be able to use 100 percent of the CPU
resources they need. If they do not use all the CPU resources, then users assigned to this resource consumer group will use the remaining CPU resources in this manner:

- 50 percent of any remaining CPU resources after Level 1 needs are serviced
- 50 percent of any remaining CPU resources after Level 2 needs are serviced
- 25 percent of any remaining CPU resources after Level 3 needs are serviced

The sum of the emphasis method percentages at any given level must be less than, or equal to, 100.

Resource Manager uses similar techniques for managing Parallel Query resources.

**FIGURE 10.5** Managing CPU resources with resource group, resource plans, and resource plan directives

```
<table>
<thead>
<tr>
<th>Resource Consumer Group</th>
<th>Power_Users</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Resource Plan</strong></td>
<td>Technical</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Resource Plan Directive</strong></td>
<td>CPU Utilization</td>
</tr>
<tr>
<td>Level 1: 100</td>
<td>Level 1: 80</td>
</tr>
<tr>
<td>Level 2: 50</td>
<td>Level 2: 60</td>
</tr>
<tr>
<td>Level 3: 50</td>
<td>Level 3: 40</td>
</tr>
<tr>
<td>Level 4: 25</td>
<td>Level 4: 40</td>
</tr>
<tr>
<td>Level 5: 0</td>
<td>Level 5: 20</td>
</tr>
<tr>
<td>Level 6: 0</td>
<td>Level 6: 10</td>
</tr>
<tr>
<td>Level 7: 0</td>
<td>Level 7: 0</td>
</tr>
<tr>
<td>Level 8: 0</td>
<td>Level 8: 0</td>
</tr>
</tbody>
</table>
```
**How the Emphasis Method Manages Parallel Query Resources**

The Parallel Query (PQ) resource that is controlled by Resource Manager is the PQ degree of parallelism. *Degree of parallelism* refers to the number of query slave processes that an individual session is allowed to start when processing its query. The more query slaves started, the faster the query will be processed. However, this increased speed comes at the expense of increased CPU utilization and disk I/O. Resource manager uses the *Absolute method* to control Parallel Query resources. This means that the maximum degree of parallelism is a specific (i.e., absolute) value. We can use the analogy of building a home to illustrate this concept.

Suppose you were an affluent homeowner who is looking to build an expensive new house and your builder employs ten carpenters. However, this builder will also be working on seven other customers’ houses at the same time your home is being constructed. The builder’s construction manager (Resource Manager) is responsible for making sure that the ten carpenters (resources) employed by the builder complete all the work required to build the homes. If several carpenters work together (i.e., *parallel*) on your home, your new home will be completed more quickly than if one carpenter worked alone. However, if all the carpenters work exclusively on your home, the other customers’ homes will not be completed on time.

Since your home is the largest (and most profitable), the construction manager has decided to allow up to four carpenters to work on your house simultaneously. The construction manager has also decided to allow up to three carpenters to work simultaneously on the second-most-profitable house under construction, and the remaining carpenters will work on the last six houses being constructed.

The previous analogy is conceptually identical to the way Resource Manager controls requests for access to the PQ resources. For example, suppose you decided to limit PQ resources for users in the resource consumer group called *POWER_USERS*. As with CPU resources, the two resource plans, TECHNICAL and FUNCTIONAL, can also be assigned PQ priority on up to eight levels of granularity. Figure 10.6 conceptually shows how this resource allocation can be broken down.

The FUNCTIONAL resource plan shown in Figure 10.6 can utilize up to five Parallel Query processes when performing Parallel Query operations. If any Parallel Query resources are left over, then the remaining Parallel Query resources will be allocated in this manner:

- Up to five additional query processes can be started if any resources remain after Level 1 needs are serviced.
- Up to two additional query processes can be started if any resources remain after Level 2 needs are serviced.
- One additional query process can be started for each of the remaining levels if any resources remain from the previous level.

**Figure 10.6** Managing Parallel Query resources with resource groups, resource plans, and resource plan directives

Resource Manager can only manage the resources (i.e., CPUs) that are allocated to the instance by the server’s operating system. If other OS resource management tools are already managing the allocation of resources to Oracle at the OS level, then Resource Manager’s effectiveness may be limited.

The CPU and Parallel Query resource limits do not affect processing unless these resources are scarce at the system level. For example, if excess CPU resources exist on your server, then CPU resource management will have no effect. This would allow a process assigned little or no resources via a Plan Directive to be able to acquire as many CPU resources as it needs—even more than the amount specified in its assigned directive.
Managing Undo Resources

Oracle9i introduces a new Plan Directive called the UNDO_PLAN. Using this Plan Directive you can control the amount of undo segment space a user is allowed to consume when executing a large DML transaction. The limits placed on undo segment usage are implemented in the form of a quota for the entire resource group. If the quota on undo segment space is exceeded, the user performing the UPDATE which caused the quota to be exceeded will have their UPDATE statement terminated. Additional DML cannot be performed by the user, or any other member of the user’s resource group, until another member of the resource group releases undo resources.

Managing Workload Resources

Through a mechanism called the Active Session Pool, Oracle9i allows you to limit the number of active concurrent sessions for a particular resource group. Once the members of the resource group reach the number of allowed active concurrent sessions, any subsequent requests for active sessions will be queued by Resource Manager instead of being processed immediately. In this way, the Active Session Pool allows you to guarantee a minimum number of resources to each resource group, without allowing an individual resource group to monopolize the server’s available resources. Any queued sessions will be executed after one or more of the existing sessions completes or becomes inactive. There is one queue per resource consumer group. Queued sessions are allocated resources on a first-in first-out basis.

Non-zero values in the CURRENT_QUEUE_DURATION column of the V$SESSION dynamic performance view indicate that a session is currently queued by Resource Manager. The QUEUE_LENGTH column of the V$RSRC_CONSUMER_GROUP also contains useful information for monitoring the total number of queued sessions for a particular resource group.

Both the maximum number of allowed sessions and the maximum number of queued sessions can be specified when the resource group is created. The maximum number of sessions is specified using the ACTIVE_SESS_POOL_P1 Resource Manager parameter. The maximum time, in seconds, that a session will be queued before aborting is specified using the QUEUEING_P1 Resource Manager parameter. The default value for both the ACTIVE_SESS_POOL_P1
and QUEUEING_P1 parameters is 1,000,000. This means that a maximum of 1,000,000 sessions are allowed for each resource group and each session can be queued for up to 1,000,000,000 seconds before aborting.

A session is considered active if it is currently participating in a DML transaction, performing a query, or some other parallel operation.

**Managing Execution Duration**

Using the new Oracle9i resource plan directive called MAX_ESTIMATED_EXEC_TIME, you can specify the maximum length of time that a database operation can take to complete. Once set, Resource Manager will use the statistics in the CBO to generate an estimated completion time for each database operation. This estimate is then compared to this specified maximum allowed time. If the operation is going to take longer than the specified duration, then the operation will not be started.

If a resource consumer group has several competing maximum execution time directives assigned to it, then the shortest maximum time will be used.

Now that you have a sense of how resource groups, resource plans, and resource plan directives can be used to manage CPU, Parallel Query, session, undo, and execution time resources, the following section explains the mechanics of implementing these Resource Management features.

**Configuring Resource Management**

In order to use the resource management features of Oracle9i, you must first perform a few simple configuration operations. These operations include granting resource management privileges to the proper users, creating the resource management objects, assigning those resources to individual users, and setting the overall resource objectives for the instance.
Granting Resource Manager Privileges

Users who are going to be allowed to manage Resource Groups, Plans, and Directives must be granted the ADMINISTER_RESOURCE_MANAGER privilege. Unlike traditional system privileges, which can be granted with the GRANT ... TO ... command, the ADMINISTER_RESOURCE_MANAGER privilege must be granted using an Oracle-supplied PL/SQL package called DBMS_RESOURCE_MANAGER_PRIVS. This package requires three arguments: GRANTEE_NAME, PRIVILEGE_NAME, and ADMIN_OPTION. GRANTEE_NAME is the name of the Oracle user being granted the privilege. PRIVILEGE_NAME is the name of the privilege being granted (i.e., DBMS_RESOURCE_MANAGER_PRIVS). ADMIN_OPTION indicates whether the user receiving the privilege can pass that privilege along to other users. Figure 10.7 shows how this package could be used to grant the ADMINISTER_RESOURCE_MANAGER privilege to the user Regi with the ADMIN option.

**FIGURE 10.7** Granting Resource Manager privilege

```sql
SQL> EXECUTE dbms_resource_manager_privs.grant_system_privilege (grantee_name => 'REGI', privilege_name => 'ADMINISTER/resource_manager_privs', admin_option => TRUE);
PL/SQL procedure successfully completed.
```

The => in Figure 10.7 is part of the actual syntax for executing the DBMS_RESOURCE_MANAGER_PRIVS package. You must also type all the parameters on one line, as shown in Figure 10.7, in order for the package to execute successfully.

If you need to revoke the privilege to administer Resource Manager, you will use the REVOKE_SYSTEM_PRIVILEGE procedure in the DBMS_RESOURCE_MANAGER_PRIVS package. This procedure accepts two arguments: REVOKEE_NAME and PRIVILEGE_NAME. Figure 10.8 shows how the Resource Manager privilege could be revoked from the user Regi.

**FIGURE 10.8** Revoking Resource Manager privilege

```sql
SQL> EXECUTE dbms_resource_manager_privs.revoke_system_privilege (revokee_name => 'REGI', privilege_name => 'ADMINISTER/resource_manager_privs');
PL/SQL procedure successfully completed.
```
Once a user has been assigned the `ADMINISTER_RESOURCE_MANAGER` privilege, the user can then create other Resource Manager objects.

**Creating the Resource Manager Objects**

Once the proper privileges have been assigned to the users who will be administering the Resource Manager, you must then build the database objects needed by Resource Manager. Setting up Resource Manager requires the following steps:

- Creation of a pending area
- Creation of one or more resource consumer groups
- Creation of one or more resource plans
- Creation of one or more resource plan directives
- Validation of the resource consumer groups, plans, and directives
- Saving the resource consumer groups, plans, and directives to the database

**Creating the Pending Area**

Whenever a new resource consumer group, plan, or directive is created, it is temporarily stored in the pending area until it is validated and written to the database. The purpose of the pending area is to give the DBA an opportunity to confirm that the definition of each consumer group, plan, and directive is correct before implementing it. The pending area is created using the `DBMS_RESOURCE_MANAGER` package and the `CREATE_PENDING_AREA` procedure:

```
SQL> EXECUTE dbms_resource_manager.create_pending_area;
```

The database can have only one pending area at a time.
Creating Resource Consumer Groups

A resource consumer group (RCG) is used to define a group of application users who have similar resource requirements. RCGs are similar to Oracle roles, except RCGs are used to manage resources, whereas roles are used to manage database object and system privileges. By default, every application user belongs to at least one RCG, which is called the DEFAULT_CONSUMER_GROUP. In addition to the DEFAULT_CONSUMER_GROUP, several other RCGs are constructed at database creation. The names and descriptions of these RCGs are shown in Table 10.7.

<table>
<thead>
<tr>
<th>Resource Group Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYS_GROUP</td>
<td>This group has the highest priority for resource access. The users SYS and SYSTEM are assigned to this RCG by default.</td>
</tr>
<tr>
<td>LOW_GROUP</td>
<td>This group receives the lowest priority for database resources. Every user has access to this RCG.</td>
</tr>
<tr>
<td>DEFAULT_CONSUMER_GROUP</td>
<td>Every user who is not explicitly assigned to another RCG is assigned to this RCG by default.</td>
</tr>
<tr>
<td>OTHER_GROUPS</td>
<td>The group to which all users belong when they are not members of the specific resource plan that is currently active in the instance.</td>
</tr>
</tbody>
</table>

If you want to create your own RCG, you will use the DBMS_RESOURCE_MANAGER package and the CREATE_CONSUMER_GROUP procedure. Figure 10.9 shows this package being used to create a RCG called POWER_USERS.

At this point, the new POWER_USERS resource consumer group is stored in the pending area, which you created in the previous section. The new RCG will
not be written to the database and will not be made available until it is committed. Committing new RCGs to the database is covered later in this chapter.

### Oracle Objective

Create resource plans within groups

### Creating Resource Plans

Resource plans are used to organize plan directives. A group of plan directives can be assigned to a resource plan, which in turn can be assigned to one or more resource groups. Only one resource plan can be active in the instance at a time. Determining which resource plan is active in the instance is discussed in the “Setting Instance Resource Objectives” section later in this chapter.

Suppose you wanted to create two resource plans within the POWER_USERS resource consumer group, one for the functional users and one for the technical users. Figure 10.10 shows how these two resource plans could be created.

**FIGURE 10.10** Creating new resource plans

```sql
SQL> EXECUTE dbms_resource_manager.create_plan (plan => 'FUNCTIONAL', comment => 'Functional power users');
PL/SQL procedure successfully completed.

SQL> EXECUTE dbms_resource_manager.create_plan (plan => 'TECHNICAL', comment => 'Technical power users');
PL/SQL procedure successfully completed.
```

Three default resource plans, called SYSTEM_PLAN, INTERNAL_PLAN, and INTERNAL QUIESCE, are created at database creation.

As with the creation of a new resource consumer group, the new FUNCTIONAL and TECHNICAL resource plans are stored in the pending area until they are committed. Committing new resource plans to the database is also covered in later in this chapter.

**NOTE**

A resource plan can be assigned to other plans as well as resource consumer groups. These are referred to as subplans. When subplans are used, the maximum allowed degree of parallelism would be the minimum of all the possible subplan values.
Creating Plan Directives

A plan directive is used to link a resource consumer group to a resource plan. The plan directive is where the appropriate allocation of the resources controlled by Resource Manager (e.g., CPU, degree of parallelism, etc.) is specified. Figure 10.11 shows the assignment of CPU and parallelism resources to the FUNCTIONAL and TECHNICAL resource groups in the POWER_USERS resource consumer group.

**FIGURE 10.11 Creating a new plan directive**

The CPU_P1 argument shown in Figure 10.11 is used to specify what percentage of the overall CPU the highest priority resource consumer group/resource plan combination is allowed to use. As described earlier, you can also define up to eight levels of resource allocation for both CPU and Parallel Query using CPU_P1 and parallel_degree_limit_p1, CPU_P2 and parallel_degree_limit_p2, etc., through CPU_P8 and parallel_degree_limit_p8. Each plan directive must also include a reference to the default resource consumer group OTHER_GROUPS. This is the resource consumer group that is active for a user when the user is not included in the resource plan that is active at the instance level. Figure 10.12 shows the assignment of FUNCTIONAL and TECHNICAL resource plans to the OTHER_GROUPS resource consumer group.

**FIGURE 10.12 Allocating a plan directive to OTHER_GROUPS**

Validating Consumer Resource Group, Plans, and Directives

Once a resource consumer group has been defined and assigned a resource plan and directive, the entire package must be validated before being written.
to the database. Verification of these Resource Manager components is performed using the `VALIDATE_PENDING_AREA` procedure of the `DBMSRESOURCE_MANAGER` package as shown here:

```
SQL> EXECUTE dbms_resource_manager.validate_pending_area ();
```
If you forget to include a reference to the `OTHER_GROUPS` default resource consumer group when defining the `FUNCTIONAL` and `TECHNICAL` plan directives, you will see the error shown in Figure 10.13 when validation is performed.

**FIGURE 10.13** Error returned when `OTHER_GROUPS` is omitted from the plan directive

<table>
<thead>
<tr>
<th>SQL&gt; execute dbms_resource_manager.validate_pending_area;</th>
</tr>
</thead>
<tbody>
<tr>
<td>BEGIN dbms_resource_manager.validate_pending_area; END;</td>
</tr>
<tr>
<td><strong>ERROR at line 1:</strong></td>
</tr>
<tr>
<td>ORA-00004: error occurred at recursive SQL level</td>
</tr>
<tr>
<td>ORA-01001: invalid cursor</td>
</tr>
<tr>
<td>ORA-25382: validation of pending area failed</td>
</tr>
<tr>
<td>ORA-25377: consumer group <code>OTHER_GROUPS</code> is not part of top-plan <code>FUNCTIONAL</code></td>
</tr>
<tr>
<td>ORA-25377: consumer group <code>OTHER_GROUPS</code> is not part of top-plan <code>FUNCTIONAL</code></td>
</tr>
<tr>
<td>ORA-06512: at &quot;SYS.UDDE_Msg&quot;, line 249</td>
</tr>
<tr>
<td>ORA-06512: at &quot;SYS.UDDE_Msg&quot;, line 254</td>
</tr>
<tr>
<td>ORA-06512: at line 1</td>
</tr>
</tbody>
</table>

Some of the other errors that can also cause validation errors include these:

- Giving a resource consumer group and resource plan identical names
- Attempting to create more than 32 resource consumer groups within any active plan
- Having CPU allocation percentages that sum to greater than 100 percent for one level
- Creating a loop via the use of subplans
- Attempting to delete a plan that is the topmost active plan in the instance

**Saving Resource Consumer Groups, Plans, and Directives**

Once the pending resource consumer groups, plans, and directives have been validated, they must be committed to the database before they can be assigned to database users. This commit is performed using the `SUBMIT_PENDING_AREA` procedure in the `DBMSRESOURCE_MANAGER` package as shown here:

```
SQL> EXECUTE dbms_resource_manager.submit_pending_area;
```
Once you have created resource consumer groups, assigned them resource plans, and associated resource directives with those plans, it is time to assign the resource consumer groups to application users.
Assigning Resources to Users

You can assign a resource consumer group to a database user by either granting the RCG directly to an individual user, or by granting the RCG to a database role. Figure 10.14 shows how the `GRANT_SWITCH_CONSUMER_GROUP` procedure in the `DBMS_RESOURCE_MANAGER_PRIVS` package is used to assign the `POWER_USERS` RCG to a user named Betty, without the ability to pass the privileges along to other users (i.e., lacking `GRANT OPTION`).

**Figure 10.14** Assigning a resource consumer group to a user

```sql
EXECUTE dbms_resource_manager_privs.grant_switch_consumer_group (grantee_name => 'BETTY', consumer_group => 'POWER_USERS', grant_option => FALSE);
PL/SQL procedure successfully completed.
```

Since a user can be a member of several RCGs at one time, it is usually preferred to specify a default RCG that will be active for the user when they initially connect to the database. A user's default RCG is assigned using the `SET_INITIAL_CONSUMER_GROUP` procedure of the `DBMS_RESOURCE_MANAGER` package. Figure 10.15 shows how Betty would be assigned the default RCG of `POWER_USERS`. If a user is not assigned a default consumer group, their initial RCG will be the public group `DEFAULT_CONSUMER_GROUP`.

**Figure 10.15** Assigning a user a default resource consumer group

```sql
EXECUTE dbms_resource_manager.set_initial_consumer_group (user => 'BETTY', consumer_group => 'POWER_USERS ');
PL/SQL procedure successfully completed.
```

While RCGs can be assigned via roles, a resource consumer group that was granted through a role cannot be specified as a user’s default RCG.

The active RCG for a user can also be changed dynamically with the application by using the `DBMS_SESSION` package and the `SWITCH_CURRENT_CONSUMER_GROUP` procedure. This is a useful feature that allows an application to activate and deactivate various RCGs for users as they navigate through an application and perform activities that have varying levels of resource...
requirements. Figure 10.16 shows sample PL/SQL code that could be embedded in an application to use DBMS_SESSION to change a user’s current RCG to POWER_USERS when performing a resource-intensive calculation.

**FIGURE 10.16** Using DBMS_SESSION to change the active resource consumer group

```sql
CREATE OR REPLACE PROCEDURE calc_tax IS
  v_original_reg varchar2(30);
  v_previous_reg varchar2(30);
  CURSOR c_sales IS
  SELECT sales_amount FROM sales;
  v_tax number;
BEGIN
  -- set the user's RCG to POWER_USERS before calculating the tax
  DBMS_SESSION.SWITCH_CURRENT_CONSUMER_GROUP (new_consumer_group => "POWER_USERS", old_consumer_group => v_original_reg, initial_group_on_error => FALSE);
  FOR r_sales IN c_sales LOOP
    v_tax := r_sales.sales_amount * .065;
    dbms_output.put_line (v_tax);
  END LOOP;
  -- switch back to the original RCG
  DBMS_SESSION.SWITCH_CURRENT_CONSUMER_GROUP (new_consumer_group => v_original_reg, old_consumer_group => v_previous_reg, initial_group_on_error => TRUE);
END;
/
```

Once all the resource management groups, plans, and directives are in place and have been assigned to users, the Resource Management feature must be enabled for the directives to take effect.

**Automatic Consumer Group Switching**

If a user is active for a specified period of time, Resource Manager can be configured to automatically switch that user’s session to a secondary resource group. This feature is managed by specifying three resource plan directive parameters:

- **SWITCH_TIME**
- **SWITCH_GROUP**
- **SWITCH_ESTIMATE**
Setting Instance Resource Objectives

Any of the defined resource plans can be activated for the instance. However, only one resource plan can be active at a time. The active resource plan for the instance can be defined using the RESOURCE_MANAGER_PLAN parameter in init.ora, or by dynamically issuing the ALTER SYSTEM SET RESOURCE_MANAGER_PLAN command to specify which resource plan is active for the instance. If this parameter is not set, then the features of Resource Manager are disabled. Figure 10.17 shows how the instance-level resource plan could dynamically be set to the TECHNICAL plan within the POWER_USERS resource consumer group.

**FIGURE 10.17** Setting the instance’s resource plan dynamically

```
SQL> alter system set resource_manager_plan = technical;
System altered.
```

If you specify a nonexistent resource plan in init.ora, Oracle will return an ORA-07452 “Specified Resource Manager Plan Does Not Exist in the Data Dictionary Error at Instance Startup” error message.

Oracle also gives you the ability to dynamically change the active RCG for an individual connected user. Figure 10.18 demonstrates how the POWER_USERS RCG could be activated in Betty’s session by using the SWITCH_CONSUMER_GROUP_FOR_USER procedure in the DBMSRESOURCE_MANAGER package.
FIGURE 10.18  Changing the active resource consumer group for a connected user

Any changes made to the active resource plan will take effect immediately.

In some applications, the application connects to the database with the same database username for all users (even though each application user has a unique login name). In these cases, the technique shown in Figure 10.18 cannot be used since all connected users would then have the same RCG activated. In this situation, you can use the SWITCH_CONSUMER_GROUP_FOR_SESS procedure in the DBMS_RESOURCE package to change the active RCG for an individual user’s session—even if that user is connected to the database with the same username as another connected user. This procedure accepts the SID and SERIAL# of the session you wish to change as input parameters. Both SID and SERIAL# are identified by querying V$SESSION. Figure 10.19 demonstrates how this procedure could be used to change the RCG for just Betty’s session when all application users connect to the database using the database username APPS.

FIGURE 10.19  Changing the active resource consumer group for a specified session

If they have been granted the privileges allowing them to switch to that particular consumer group, non-DBA users can also change their active resource consumer group by using the SWITCH_CURRENT_CONSUMER_GROUP procedure in the DBMS_SESSION package.
Determining Application User CPU Utilization

You've decided that you'd like to explore the possibility of using the Resource Management features of Oracle9i to control the amount of CPU resources each application user can consume. So how do you go about deciding how much CPU you should allow each application user's resource group to consume?

One way to determine how many CPU resources users are consuming is to compare the CPU usage for a particular user session to the CPU usage for the entire instance. You can do this by using the following script shown here:

```
SET echo OFF
SET feedback OFF
SET verify OFF
SET SERVEROUT ON
ACCEPT username PROMPT 'Enter the username of the user whose CPU ratio you wish to calculate: '
DECLARE
  v_username VARCHAR2(30);
  v_session_cpu NUMBER:=0;
  v_system_cpu NUMBER:=0;
  v_cpu_ratio NUMBER:=0;
BEGIN
  v_username := UPPER(username);
  SELECT value
  INTO v_session_cpu
  FROM v$session
  WHERE sid = (SELECT sid
              FROM v$session
              WHERE username = v_username)
  AND statistic# = 12;
  SELECT value
  INTO v_system_cpu
  FROM v$sysstat
  WHERE name = 'CPU used by this session';
  v_cpu_ratio := ROUND((v_session_cpu/v_system_cpu)*100,2);
  dbms_output.put_line ('v_cpu_ratio ' || v_cpu_ratio || ' % of the total system CPU used was requested by the user ' || v_username);
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    dbms_output.put_line ('The user entered is not currently connected');
END;
```

When executed, this script will prompt you for the name of the connected user whose CPU consumption you would like to calculate:

```
SQL> @userCPURatio.sql
Enter the username of the user whose CPU ratio you wish to calculate: BETTY
25.52% of the total system CPU used was requested by the user BETTY.
```

The output from the query indicates that the user BETTY has consumed approximately 25 percent of the total CPU calls that have been made since
instance startup. Therefore, any RCG that the user BETTY is assigned to should have CPU allocation of 25 percent at level 1 in its resource plan directive in order for her to be able to continue to enjoy the performance she is accustomed to.

Monitoring Resource Manager

Once resource consumer groups, resource plans, and plan directives have been created, you can use several places to monitor Resource Manager–related topics, such as which consumer groups exist, which plan is active, and which plans have been assigned to each user. These places include dynamic performance views, DBA data dictionary views, and Oracle Enterprise Manager.

Resource Manager Dynamic Performance Views

The three dynamic performance views that contain information related to Resource Manager are:

- V$SESSION
- V$RSRC_CONSUMER_GROUP
- V$RSRC_PLAN

The two other V$ views related to Resource Manager are V$RSRC_PLAN_CPU_MTH and V$RSRC_CONSUMER_GROUP_CPU_MTH. These two views contain only one column, NAME, which displays the available resource-allocation methods for both resource consumer groups and resource plans, respectively. By default, the methods used are EMPHASIS and ROUND-ROBIN, respectively.

V$SESSION

The RESOURCE_CONSUMER_GROUP column in V$SESSION displays the current RCG for each session. The following query on V$SESSION shows the contents of this column:

```sql
SQL> SELECT username, resource_consumer_group
2   FROM v$session
3  WHERE username IS NOT NULL;
```
V$RSRC_CONSUMER_GROUP

The V$RSRC_CONSUMER_GROUP dynamic performance view contains details about the resource allocations that have been made to the active resource consumer groups. Table 10.8 describes the contents of this view.

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAME</td>
<td>The name of the resource consumer group</td>
</tr>
<tr>
<td>ACTIVE_SESSIONS</td>
<td>The number of sessions that are currently assigned to this resource consumer group</td>
</tr>
<tr>
<td>EXECUTION_WAITERS</td>
<td>The number of sessions assigned to this resource consumer group that are waiting to be allocated resources before they can execute</td>
</tr>
<tr>
<td>REQUESTS</td>
<td>The total number of database requests that have been executed by users in this resource consumer group</td>
</tr>
<tr>
<td>CPU_WAIT_TIME</td>
<td>The total time that users in this resource consumer group have had to wait for access to the CPU (in 100ths of a second)</td>
</tr>
<tr>
<td>CPU_WAITS</td>
<td>The total number of times that users in the resource consumer group have waited for access to the CPU</td>
</tr>
<tr>
<td>CONSUMED_CPU_TIME</td>
<td>The total amount of CPU time that users in this resource consumer group have consumed (in 100ths of a second)</td>
</tr>
<tr>
<td>YIELDS</td>
<td>The number of times users in this resource consumer group have had to yield the CPU to users in other resource consumer groups with greater priority</td>
</tr>
</tbody>
</table>
Ideally, if your users have been assigned to resource consumer groups effectively, users with high priority in the database should be experiencing few CPU waits (EXECUTION_WAITERS and CPU_WAIT_TIME) and not have many unserviced sessions (SESSIONS_QUEUED). If this is not the case, you may want to reexamine the resource allocations you have created to try and minimize these events.

**V$RSRC_PLAN**

The V$RSRC_PLAN dynamic performance view displays the names of all the currently active resource plans. There is only one column in this view, NAME, which contains the name of the resource plan that is active at the instance-level. The following query shows how V$RSRC_PLAN could be used to determine the active resource plan:

```sql
SQL> SELECT *
   2  FROM v$rsrc_plan;
```

<table>
<thead>
<tr>
<th>NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>TECHNICAL</td>
</tr>
</tbody>
</table>

The output from this query shows that the TECHNICAL resource plan is currently active in the database. This can be either because the init.ora parameter RESOURCE_MANAGER_PLAN is set to TECHNICAL, or because the ALTER SYSTEM SET RESOURCE_MANAGER_PLAN=’TECHNICAL’ command was issued.

### Resource Manager Data Dictionary Views

There are six data dictionary views that contain information related to Resource Manager:

- DBA_RSRC_CONSUMER_GROUPS
- DBA_RSRC_CONSUMER_GROUPS_PRIVS
- DBA_RSRC_PLANS

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SESSIONS_QUEUED</td>
<td>The number of user sessions assigned to this resource consumer group that are waiting to become active</td>
</tr>
</tbody>
</table>
• DBA_RSRC_PLAN_DIRECTIVES
• DBA_RSRC_MANAGER_SYSTEM_PRIVS
• DBA_USERS

**DBA_RSRC_CONSUMER_GROUPS**
The DBA_RSRC_CONSUMER_GROUPS view contains information on each resource consumer group in the database. These are the same parameters that are specified at the creation of the resource consumer group. Table 10.9 lists the contents of this view.

**Table 10.9** Contents of the **DBA_RSRC_CONSUMER_GROUPS** View

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONSUMER_GROUP</td>
<td>The name of the resource consumer group.</td>
</tr>
<tr>
<td>CPU_METHOD</td>
<td>The CPU allocation method used to assign CPU resources to this consumer group.</td>
</tr>
<tr>
<td>COMMENTS</td>
<td>The comments (if any) specified for the resource consumer group at its creation.</td>
</tr>
<tr>
<td>STATUS</td>
<td>The status of the resource consumer group. New, uncommitted RCGs have a status of PENDING. Committed RCGs have a status of ACTIVE.</td>
</tr>
<tr>
<td>MANDATORY</td>
<td>Indicates whether or not the resource consumer group is mandatory. Mandatory RCGs cannot be deleted or modified.</td>
</tr>
</tbody>
</table>

The following query on DBA_RSRC_CONSUMER_GROUPS shows how the summary information for the POWER_USERS resource consumer group created earlier can be examined:

```sql
SQL> SELECT consumer_group, cpu_method,
          status, mandatory
FROM dba_rsrc_consumer_groups
WHERE consumer_group = 'POWER_USERS';
```
CONSUMER_GROUP   CPU_METHOD STATUS    MAN
-------------- ------------ ------- ----
POWER_USERS     ROUND-ROBIN ACTIVE NO

**DBA_RSRC_CONSUMER_GROUP_PRIVS**

The DBA_RSRC_CONSUMER_GROUP_PRIVS view shows which users or roles in the database have been granted to each resource consumer group. Table 10.10 lists the columns found in this view.

**TABLE 10.10** Contents of the DBA_RSRC_CONSUMER_GROUP_PRIVS View

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRANTEE</td>
<td>The username or role that has been granted a resource consumer group</td>
</tr>
<tr>
<td>GRANTED_GROUP</td>
<td>The name of the resource consumer group the user or role has been granted</td>
</tr>
<tr>
<td>GRANT_OPTION</td>
<td>Indicates whether the user has the ability to pass the privilege, assigned via the resource consumer group, on to other users</td>
</tr>
<tr>
<td>INITIAL_GROUP</td>
<td>Indicates whether the resource consumer group is the user’s default RCG</td>
</tr>
</tbody>
</table>

The following query shows the resource consumer group privileges for the two users, Apps and Betty, which were referenced in prior examples:

```
SQL> SELECT * 
2  FROM dba_rsrb_consumer_group_privs;
```

<table>
<thead>
<tr>
<th>GRANTEE</th>
<th>GRANTED_GROUP</th>
<th>GRA</th>
<th>INI</th>
</tr>
</thead>
<tbody>
<tr>
<td>APPS</td>
<td>POWER_USERS</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>BETTY</td>
<td>POWER_USERS</td>
<td>NO</td>
<td>YES</td>
</tr>
<tr>
<td>PUBLIC</td>
<td>DEFAULT_CONSUMER_GROUP</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>PUBLIC</td>
<td>LOW_GROUP</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>SYSTEM</td>
<td>SYS_GROUP</td>
<td>NO</td>
<td>YES</td>
</tr>
</tbody>
</table>
The `DBA_RSRC_PLANS` view shows all the resource plans that have been created in the database. Table 10.11 shows the contents of this view.

**TABLE 10.11** Contents of the `DBA_RSRC_PLANS` View

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLAN</td>
<td>The name of the plan.</td>
</tr>
<tr>
<td>NUM_PLAN_DIRECTIVES</td>
<td>The number of directives within the plan.</td>
</tr>
<tr>
<td>CPU_METHOD</td>
<td>The method used to allocate CPU resources to the plan (Emphasis method).</td>
</tr>
<tr>
<td>ACTIVE_SESS_POOL_MTH</td>
<td>Allocation method used for Active Session Pool resources.</td>
</tr>
<tr>
<td>PARALLEL_DEGREE_LIMIT_MTH</td>
<td>The method used to allocate parallel resources to the plan (Absolute method).</td>
</tr>
<tr>
<td>QUEUING_MTH</td>
<td>Allocation method used for Queuing session resources.</td>
</tr>
<tr>
<td>COMMENTS</td>
<td>The comments defined at plan creation (if any).</td>
</tr>
<tr>
<td>STATUS</td>
<td>The status of the resource plan. New uncommitted plans have a status of PENDING. Committed plans have a status of ACTIVE.</td>
</tr>
<tr>
<td>MANDATORY</td>
<td>Indicates whether or not the resource plan is mandatory. Mandatory plans cannot be modified or deleted.</td>
</tr>
</tbody>
</table>

The following query shows the output from `DBA_RSRC_PLANS` for the TECHNICAL and FUNCTIONAL resource plans created in a previous example. It also shows an uncommitted resource plan called AR_CLERK:

```
SQL> SELECT plan, status, mandatory
2  FROM dba_rsrtc_plans;
```
Understanding Resource Manager

<table>
<thead>
<tr>
<th>PLAN</th>
<th>STATUS</th>
<th>MAN</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSTEM_PLAN</td>
<td>ACTIVE</td>
<td>NO</td>
</tr>
<tr>
<td>FUNCTIONAL</td>
<td>ACTIVE</td>
<td>NO</td>
</tr>
<tr>
<td>TECHNICAL</td>
<td>ACTIVE</td>
<td>NO</td>
</tr>
<tr>
<td>AR_CLERK</td>
<td>PENDING</td>
<td>NO</td>
</tr>
</tbody>
</table>

**DBA_RSRC_PLAN_DIRECTIVES**

DBA_RSRC_PLAN_DIRECTIVES displays the specific resource allocations that were assigned to each resource plan. Table 10.12 shows descriptions of some of the columns found in this view.

**TABLE 10.12** Partial Contents of the **DBA_RSRC_PLAN_DIRECTIVES** View

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLAN</td>
<td>The name of the plan to which this directive has been assigned.</td>
</tr>
<tr>
<td>CPU_P1</td>
<td>The emphasis percentage assigned to CPU allocation at each level. There are seven other columns (CPU_P2 through CPU_P8) that are also used to report the plan’s CPU allocation at those levels.</td>
</tr>
<tr>
<td>COMMENTS</td>
<td>The comments defined at plan creation (if any).</td>
</tr>
<tr>
<td>STATUS</td>
<td>The status of the resource plan. New, uncommitted plans have a status of PENDING. Committed plans have a status of ACTIVE.</td>
</tr>
<tr>
<td>MANDATORY</td>
<td>Indicates whether or not the resource plan is mandatory. Mandatory plans cannot be modified or deleted.</td>
</tr>
</tbody>
</table>

The following query shows the output from DBA_RSRC_PLAN_DIRECTIVES for the plan directives created in previous examples:

```sql
QL> SELECT plan, cpu_p1, cpu_p2, cpu_p3, cpu_p4, status, mandatory
FROM dba_rsric_plan_directives;
```
### PLAN

<table>
<thead>
<tr>
<th>PLAN</th>
<th>CPU_P1</th>
<th>CPU_P2</th>
<th>CPU_P3</th>
<th>CPU_P4</th>
<th>STATUS</th>
<th>MAN</th>
</tr>
</thead>
<tbody>
<tr>
<td>TECHNICAL</td>
<td>100</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>ACTIVE</td>
<td>NO</td>
</tr>
<tr>
<td>FUNCTIONAL</td>
<td>80</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>ACTIVE</td>
<td>NO</td>
</tr>
<tr>
<td>TECHNICAL</td>
<td>0</td>
<td>50</td>
<td>0</td>
<td>0</td>
<td>ACTIVE</td>
<td>NO</td>
</tr>
<tr>
<td>FUNCTIONAL</td>
<td>0</td>
<td>80</td>
<td>0</td>
<td>0</td>
<td>ACTIVE</td>
<td>NO</td>
</tr>
<tr>
<td>TECHNICAL</td>
<td>0</td>
<td>0</td>
<td>50</td>
<td>0</td>
<td>ACTIVE</td>
<td>NO</td>
</tr>
<tr>
<td>FUNCTIONAL</td>
<td>0</td>
<td>0</td>
<td>80</td>
<td>0</td>
<td>ACTIVE</td>
<td>NO</td>
</tr>
<tr>
<td>TECHNICAL</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>25</td>
<td>ACTIVE</td>
<td>NO</td>
</tr>
<tr>
<td>FUNCTIONAL</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>20</td>
<td>ACTIVE</td>
<td>NO</td>
</tr>
</tbody>
</table>

**DBA_RSRC_MANAGER_SYSTEM_PRIVS**

The DBA_RSRC_MANAGER_SYSTEM_PRIVS view contains information about which users have been granted the ADMINISTER_RESOURCE_MANAGER privilege. Table 10.13 lists the columns found in this view.

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRANTEE</td>
<td>User to whom the ADMINISTER_RESOURCE_MANAGER privilege has been granted.</td>
</tr>
<tr>
<td>PRIVILEGE</td>
<td>The system privilege that was granted to the user.</td>
</tr>
<tr>
<td>ADMIN_OPTION</td>
<td>Indicates whether the user has the ability to grant the privilege to other users.</td>
</tr>
</tbody>
</table>

**DBA_USERS**

The DBA_USERS view has a single column, INITIAL_RSRC_CONSUMER_GROUP, which indicates the resource consumer group that has been assigned to a user as their default. The following query shows the initial resource consumer group assigned to Betty in a previous example:

```sql
SQL> SELECT username, initial_rscc_consumer_group
2  FROM dba_users
3  WHERE username = 'BETTY';
```
### Resource Manager OEM Tools

The Instance section under the Database section of the OEM Console has GUI tools that provide information on resource consumer groups and resource plans. Figure 10.20 shows the OEM resource consumer group tool.

**FIGURE 10.20** OEM resource consumer group GUI tool

![OEM resource consumer group GUI tool](image)

Figure 10.20 shows that five resource consumer groups exist in the database: `DEFAULT_CONSUMER_GROUP`, `LOW_GROUP`, `OTHER_GROUPS`, `POWER_USERS`, and `SYS_GROUP`. Figure 10.21 shows how the same GUI tool could be used to create a new resource consumer group called `FUNCTIONAL`.

The OEM Console also has a GUI tool for the management of resource plans. Figure 10.22 shows the OEM resource plan tool.
FIGURE 10.21 Creating a new resource consumer group using OEM

FIGURE 10.22 OEM resource plan GUI tool
Figure 10.22 shows that five resource plans exist in the database: FUNCTIONAL, INTERNAL_PLAN, INTERNAL_QUIESCE, SYSTEM_PLAN, and TECHNICAL. Of these, only the INTERNAL_PLAN and INTERNAL_QUIESCE are designated as mandatory. The OEM Console can also be used to create a new resource plan. Figure 10.23 shows how a new resource plan called BATCH can be created using OEM.

**FIGURE 10.23** Creating a new resource plan using OEM

---

**Summary**

Oracle Databases and Instances reside on servers. These servers have a limited amount of available CPU, memory, and disk resources. DBAs should be familiar with the utilities available on their OS for monitoring
these same resources. Any database tuning effort should also include an examination of these OS resources so that changes made to improve database performance do not cause OS-related problems like OS paging and swapping, or excessive CPU consumption.

Oracle Resource Manager is used to allocate CPU and Parallel Query resources to users. The resource plan directives are where the actual CPU and degree-of-parallelism allocations are defined using the DBMS_RESOURCE_MANAGER PL/SQL package. Once directives are defined, they are assigned to resource plans, which are assigned to resource consumer groups, which are in turn assigned to database users or roles. The resource plan for the instance can be defined using the RESOURCE_MANAGER_PLAN parameter in the init.ora or by issuing the ALTER SYSTEM SET RESOURCE_MANAGER_PLAN command. Several data dictionary and dynamic performance views are available to monitor and manage the resource consumer groups, resource plans, and resource plan directives.

Exam Essentials

**Understand the OS tuning process.** Be able to describe the order in which the components of OS tuning should be examined. Also be able to explain how to measure the current performance of those components using operating system commands and utilities.

**Know how paging and swapping differ.** Understand when paging and swapping occurs, how to detect it when it happens, and where paged or swapped objects are stored. Be able to describe what steps can be taken to minimize paging and swapping.

**Understand the role of Resource Manager.** Be able to explain all the steps required to implement Resource Manager. Know what privileges users must be granted in order to perform Resource Manager operations. Understand which data dictionary and dynamic performance views can be used to monitor Resource Manager and the contents of each.
Before you take the exam, make sure you are familiar with the following terms:

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Absolute method</td>
<td>physical memory</td>
</tr>
<tr>
<td>Active Session Pool</td>
<td>resource consumer group</td>
</tr>
<tr>
<td>degree of parallelism</td>
<td>resource plan</td>
</tr>
<tr>
<td>Emphasis method</td>
<td>resource plan directives</td>
</tr>
<tr>
<td>kernel</td>
<td>thread</td>
</tr>
<tr>
<td>pending area</td>
<td>virtual memory</td>
</tr>
</tbody>
</table>
Review Questions

1. Which Oracle versions contain the Resource Manager feature?
   A. Oracle9i
   B. Oracle9i Lite
   C. Personal Oracle9i
   D. Oracle9i Enterprise Edition

2. A group of database users who have similar resource needs can be assigned to a:
   A. Resource plan
   B. Resource consumer group
   C. Resource plan directive
   D. Consumer resource group

3. An independent sequence of instructions that executes within a single OS process is called what?
   A. Process
   B. Task
   C. Thread
   D. Executable

4. Which of the following database resources can be assigned to a resource plan using plan directives? (Choose all that apply.)
   A. Memory
   B. Undo space
   C. Disk I/O
   D. Degree of parallelism
5. At how many levels can CPU allocation be defined?
   A. Only two.
   B. Four or less.
   C. Up to eight.
   D. The number is unlimited.

6. Which of the following Oracle-supplied PL/SQL packages is used to grant a user the ability to create and administer resource management objects and privileges?
   A. DBMS_RESOURCE_MANAGER
   B. GRANT_RESOURCE_MANAGER_PRIVS
   C. DBMS_RESOURCE_MANAGER_PRIVS
   D. DBMS_RSRC_MANAGER_PRIVS

7. The pending area is used to store new Resource Manager consumer groups, plans, or directives until they are:
   A. Committed to the database
   B. Recorded in the archive logs
   C. Assigned to a database user or role
   D. Dropped from the database

8. The process of examining the contents of the pending area and determining that the resource consumer groups, resource plans, and resource plan directives found there are defined correctly is called:
   A. Certification
   B. Validation
   C. Examination
   D. Stabilization
9. Which of the following PL/SQL packages is used to create a new resource consumer group?
   A. DBMS_RESOURCE_MANAGER
   B. DBMS_CREATE_CONSUMER_GROUP
   C. DBMSRESOURCE_GROUP
   D. DBMS_RSRC_CONSUMER_GROUP

10. When an entire process is removed from memory and written to disk in order to make room for another requested process or page this is called what?
    A. Swapping
    B. Pinning
    C. Wrapping
    D. Paging

11. Which of the following resource consumer groups is active when a user is not a member of the currently active resource plan?
    A. LOW_GROUP
    B. DEFAULT_GROUP
    C. DEFAULT_CONSUMER_GROUP
    D. OTHER_GROUPS

12. The SET_INITIAL_CONSUMER_GROUP procedure in the DBMS_RESOURCE_MANAGER PL/SQL package is used to:
    A. Set the default resource plan for the instance
    B. Set the resource plan that will be active when a user logs in
    C. Create the four default resource consumer groups
    D. Alter a connected user’s current resource plan
13. Which init.ora parameter is used to define the default resource plan at the instance level?
   A. INSTANCE_RESOURCE_PLAN
   B. RESOURCE_PLAN
   C. RESOURCE_MANAGER_PLAN
   D. RSRC_MANAGER_PLAN

14. Which of the following procedures in the DBMSRESOURCE_MANAGER package can change the active resource group for a connected user? (Choose all that apply.)
   A. SWITCH_CONSUMER_GROUP_FOR_USER
   B. SWITCH_CONSUMER_GROUP_FOR_SESS
   C. SWITCH_USER_CONSUMER_GROUP
   D. SWITCH_CURRENT_CONSUMER_GROUP

15. Certain operating systems allow you to ensure that the Oracle SGA is never paged or swapped. In these cases, what init.ora parameter can be used?
   A. LOCK_SGA=TRUE
   B. NOPAGE_SGA=TRUE
   C. SGA_LOCK=TRUE
   D. PIN_SGA=TRUE

16. One of the default resource plans created at database creation is called:
   A. DEFAULT_PLAN
   B. SYSTEM_DEFAULT
   C. SYSTEM_PLAN
   D. PLAN_DEFAULT
17. Which of the following views shows information about which users have been assigned to each resource consumer group?
   A. DBA_RSRC_CONSUMER_GROUP
   B. DBA_RSRC_CONSUMER_GROUP_PRIVS
   C. DBA_RSRC_CONSUMER_GROUPS_GRANTED
   D. DBA_RSRC_CONSUMER_GROUPS

18. Which of the following views shows the resource plan that is currently active in the instance?
   A. V$RSRC_ACTIVE_PLAN
   B. V$RSRC_CURRENT_PLAN
   C. V$RSRC_PLAN
   D. V$RSRC_PLAN_ACTIVE

19. Which of the following views shows the CPU allocation directives that have been assigned to a particular resource plan?
   A. DBA_RSRC_PLAN_DIRECTIVES
   B. DBA_RSRC_DIRECTIVES
   C. DBA_RSRC_DIRECTIVE_CPU
   D. DBA_RSRC_CPU

20. “Servers with two or more CPUs, but one memory area. I/O, bus, and disk resources are used exclusively by this server” describes what type of server architecture?
   A. Uniprocessor
   B. MPP
   C. SMP
   D. NUMA
Answers to Review Questions


2. B. Users who share the same needs for access to database resources can be grouped together and assigned the same resource consumer group.

3. C. Threads are independent sequences of instructions that execute within a single OS process. This type of execution model is employed by Windows operating systems such as Windows 2000.

4. B, D. CPU utilization and the number of active session can also be defined using resource plan directives.

5. C. Varying degrees of CPU allocation can be defined at up to eight levels. Each succeeding level receives CPU access whenever excess CPU time is left over from the previous level.

6. C. The ADMINISTER_RESOURCE_MANAGER procedure in the DBMS_RESOURCE_MANAGER_PRIVS package can be used to assign a user the privileges needed to create and administer Resource Manager.

7. A. New resource consumer groups, resource plans, and resource plan directives are held in the pending area until they are validated and written to the database.

8. B. The procedure VALIDATE_PENDING_AREA in the DBMS_RESOURCE_MANAGER PL/SQL package must be used to validate new Resource Manager objects before they are written to the database.

9. A. The CREATE_CONSUMER_GROUP procedure of the DBMS_RESOURCE_MANAGER package is used to create new resource consumer groups.

10. A. Swapping is the processing of removing entire processes from memory and writing them to disk. Swapping is a very resource intensive operation that can adversely affect system performance. This should be avoided at all costs.

11. D. The OTHER_GROUPS resource consumer group is used to assign resource privileges to a user who is not a member of the resource plan that is currently active at the instance level.

12. B. When a user initiates a new connection to the database, the resource consumer group specified as their initial RCG will be active.
13. C. The default resource plan for an instance is determined at instance startup according to the RESOURCE_MANAGER_PLAN parameter in init.ora.

14. A, B. SWITCH_CONSUMER_GROUP_FOR_USER changes the current resource consumer group for a connected user by username. SWITCH_CONSUMER_GROUP_FOR_SESS changes the current resource consumer group for a connected user by SID and SERIAL#.

15. A. The LOCK_SGA=TRUE parameter can be used in cases where you would like to ensure the SGA is never paged or swapped. Consult your O/S documentation to see if this parameter can be utilized.

16. C. Every database includes at least one resource plan, called SYSTEM_PLAN. This plan is created when the database is created.

17. B. The DBA_RSRC_CONSUMER_GROUP_PRIVS view also contains information about whether this is the user’s initial resource consumer group.

18. C. The V$RSRC_PLAN view will show which of the available resource plans is active at the instance-level.

19. A. The DBA_RSRC_PLAN_DIRECTIVES view will show how much CPU has been allocated to each resource plan at each of the eight allocation levels.

20. C. This describes a Symmetric Multiprocessor (SMP) server.