Expert Oracle Database 10*g* Administration

Sam R. Alapati

Apress®

Expert Oracle Database 10*g* Administration

ISBN (pbk): 1-59059-451-7

Printed and bound in the United States of America 9 8 7 6 5 4 3 2 1

Lead Editor: Tony Davis
Technical Reviewer: John Watson
Development Editors: Robert Denn and Matthew Moodie
Editorial Board: Steve Anglin, Dan Appleman, Ewan Buckingham, Gary Cornell, Tony Davis, Jason Gilmore, Jonathan Hassell, Chris Mills, Dominic Shakeshaft, and Jim Sumser
Associate Publisher: Grace Wong
Project Manager: Beckie Stones and Tracy Brown Collins
Copy Edit Manager: Nicole LeClerc
Copy Editors: Andy Carroll, Marilyn Smith, and Susannah Pfalzer
Assistant Production Director: Kari Brooks-Copony
Production Editor: Ellie Fountain
Compositor: Dina Quan
Proofreaders: Lori Bring and Liz Welch
Indexer: John Collin
Interior Designer: Van Winkle Design Group
Cover Designer: Kurt Krames
Manufacturing Director: Tom Debolski

Distributed to the book trade worldwide by Springer-Verlag New York, Inc., 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax 201-348-4505, e-mail `orders-ny@springer-sbm.com`, or visit `http://www.springeronline.com`.

For information on translations, please contact Apress directly at 2560 Ninth Street, Suite 219, Berkeley, CA 94710. Phone 510-549-5930, fax 510-549-5939, e-mail `info@apress.com`, or visit `http://www.apress.com`.

The source code for this book is available to readers at `http://www.apress.com` in the Source Code section.

CHAPTER 4

Introduction to the Oracle Database 10*g* Architecture

In the first three chapters, I set the stage for working with Oracle. It's time now to learn about the fundamental structures of Oracle Database 10*g*. Oracle uses a set of logical structures called data blocks, extents, segments, and tablespaces as its building blocks. Oracle's physical database structure consists of data files and related files. Oracle memory structures and a set of database processes constitute the Oracle instance, and are responsible for actually performing all the work for you in the database.

To understand how the Oracle database works, you need to understand several concepts, including transaction processing, backup and recovery, undo and redo data, the optimization of SQL queries, and the importance of the data dictionary. Oracle's key features include the Recovery Manager, SQL*Plus and iSQL*Plus, Oracle Backup, the Oracle (job) Scheduler feature, the Database Resource Manager, and the Oracle Enterprise Manager management tool. This chapter provides an outline of the important Oracle automatic management features, as well as the sophisticated built-in performance tuning features, including the new Automatic Workload Repository, the Automatic Database Diagnostic Monitor, and the advisor-based Management Framework.

Before you delve deeply into the logical and physical structures that make up an Oracle database, however, you need to be clear about a fundamental concept—the difference between an Oracle *instance* and an Oracle *database*. It is very common for people to use the terms interchangeably, but they refer to different things altogether.

An *Oracle database* consists of files, both data files and Oracle system files. These files by themselves are useless unless you can interact with them somehow, and this requires the help of the operating system, which provides processing capabilities and resources, such as memory, to enable you to manipulate the data on the disk drives. When you combine the specific set of processes created by Oracle on the server with the memory allocated to it by the operating system, you get the *Oracle instance*.

You'll often hear people remarking that the "database is up," though what they really mean is that the "instance is up." The database itself, in the form of the set of physical files it's composed of, is of no use if the instance is not up and running. The instance performs all the necessary work for the database.

Oracle Database Structures

In discussing the Oracle database architecture, you can make a distinction between the physical and logical structures. You don't take all the data from the tables of an Oracle database and just put it on disk somewhere on the operating system storage system. Oracle uses a sophisticated logical view of the internal database structures that helps in storing and managing data properly in the

physical data files. By organizing space into logical structures and assigning these logical entities to users of the database, Oracle databases logically separate the database users (who own the database objects, such as tables) from the physical manifestations of the database (data files and so forth).

The following sections discuss the various logical and physical data structures.

The Logical Database Structures

Oracle databases use a set of logical database storage structures in order to manage the physical storage that is allocated in the form of operating system files. These logical structures, which primarily include tablespaces, segments, extents, and blocks, allow Oracle to control the use of the physical space allocated to the Oracle database.

Taken together, a set of related logical objects in a database is called a *schema*. Remember that Oracle database objects, such as tables, indexes, and packaged SQL code, are actually logical entities. Dividing a database's objects among various schemas promotes ease of management and a higher level of security.

Let's look at the logical composition of an Oracle database from the bottom up, starting with the smallest logical components and moving up to the largest entities:

- *Data blocks*: The Oracle *data block* is at the foundation of the database storage hierarchy and is the basis of all database storage in an Oracle database. A data block consists of a number of bytes of disk space in the operating system's storage system. All Oracle's space allocation and usage is in terms of Oracle data blocks.
- *Extents*: An *extent* is two or more contiguous Oracle data blocks, and this is the unit of space allocation.
- *Segments*: A *segment* is a set of extents that you allocate to a logical structure like a table or an index (or some other object).
- *Tablespaces*: A *tablespace* is a set of one or more data files, and usually consists of related segments. The data files contain the data of all the logical structures that are part of a tablespace, like tables and indexes.

The following sections explore each of these logical database structures in detail.

Data Blocks

The smallest logical component of an Oracle database is the *data block*. Data blocks are defined in terms of bytes. For example, you can size an Oracle data block in units of 2KB, 4KB, 8KB, 16KB, or 32KB (or even larger chunks), and it is common to refer to the data blocks as *Oracle blocks*.

The storage disks on which the Oracle blocks reside are themselves divided into *disk blocks,* which are areas of contiguous storage containing a certain number of bytes—for example, 4,096 or 32,768 bytes (4KB or 32KB; each kilobyte has 1,024 bytes).

How Big Should the Oracle Block Size Be?

You, as the DBA, have to decide how big your Oracle blocks should be and set the `DB_BLOCK_SIZE` parameter in your Oracle initialization file (the init.ora file). Think of the block size as the minimum unit for conducting Oracle's business of updating, selecting, or inserting data. When a user selects data from a table, the select operation will "read," or fetch, data from the database files in units of Oracle blocks.

If you choose the common Oracle block size of 8KB, your data block will have exactly 8,192 bytes. If you use an Oracle block size of 64KB (65,536 bytes), even if you just want to retrieve a name that's only four characters long, you'll have to read in the entire block of 64KB that happens to contain the four characters you're interested in.

Tip If you're coming to Oracle from SQL Server, you can think of the Oracle block size as being the same as the SQL Server page size.

As was mentioned earlier, the operating system also has a disk block size, and the operating system reads and writes information in whole blocks. Ideally, the Oracle block size should be a multiple of the disk block size; if not, you may be wasting time reading and writing whole disk blocks while only making use of part of the data on each I/O. On an HP-UX system, for example, if you set your Oracle block size to a multiple of the operating system block size, you gain 5 percent in performance.

Oracle offers the following guidelines for choosing the database block size:

- Choose a smaller block size if your rows are small and access is predominantly random.
- Choose a larger block size if the rows are small and access is mostly sequential (or random and sequential), or if you have large rows.

In Chapter 9, which discusses the creation of Oracle databases, you'll learn a lot more about Oracle database block size and the criteria for choosing an appropriate block size.

Note The Oracle block size that you should choose depends on what you're going to do with your database. For example, a small block size is useful if you're working with small rows and you're doing a lot of index lookups. Larger block sizes are useful in report applications when you're doing large table scans. If you are unsure about what block size to use, remember that Oracle recommends that you choose a block size of 8KB for most systems that process a large number of transactions. Only if you are dealing with LOBs (large objects) do you need to have a block size larger than 8KB.

Multiple Oracle Data Block Sizes

The `DB_BLOCK_SIZE` initialization parameter determines the standard block size in your Oracle database, and it can range from 2KB to 32KB. The system tablespace is always created with the standard block size, and Oracle lets you specify up to four additional nonstandard block sizes. For example, you can have 2KB, 4KB, 8KB, 16KB, and 32KB block sizes all within the same database—the reasons you might wish to do this are discussed shortly, in the "Tablespaces" section. If you choose to configure multiple Oracle block sizes, you must also configure corresponding subcaches in the buffer cache of the system global area (SGA), which is Oracle's memory allocation, as you'll learn in the "Understanding Main Memory" section of this chapter.

Multiple data block sizes aren't always necessary, and you'll do just fine in most cases with one standard Oracle block size. Multiple block sizes are useful primarily when transporting tablespaces between databases with different database block sizes.

What's Inside a Data Block?

All data blocks can be divided into two main parts: the row data portion and the free space portion. (There are also other smaller areas, such as overhead and header space for maintenance purposes.) The *row data* section of data blocks contains the data stored in the tables or their indexes. The *free space* section is the space left in the Oracle block for new data to be inserted or for existing rows in the block to be extended.

Sometimes it may be useful to find out exactly what data is in a particular block or to find out which block contains a particular piece of data. You can actually "see" what's inside a data block by

"dumping" the block contents. Oracle blocks can be dumped at the operating system level (referred to as *binary dumps*), and you can also perform Oracle-formatted block dumps.

The most common reason for performing a block dump is to investigate *block corruption,* which may be caused by operating system or Oracle software errors, hardware defects, or memory or I/O caching problems. Oracle does have tools that can help you restore data from corrupted data blocks, and you can adopt several other strategies to recover from data block corruption; you'll learn about these strategies in Chapter 16.

Let's look at what's actually in an Oracle data block. First, before you do a data dump, you need to find out which data file and data block you want to dump. Listing 4-1 shows a query that enables you to determine the file and block IDs.

Listing 4-1. *Query to Identify File and Block IDs*

```
SQL> SELECT segment_name,
     file_id,
     block_id
     FROM dba_extents
     WHERE owner = 'OE'
     AND segment_name LIKE 'ORDERS%';
SEGMENT_NAME                FILE_ID   BLOCK_ID
--------------------------- ---------- ------------------------------
ORDERS                         397      32811
SQL>
```

You can alternatively use the following query to get the same information:

```
SQL> SELECT header_file,header_block FROM dba_segments
     WHERE segment_name = 'PERSONS';
HEADER_FILE HEADER_BLOCK
----------- ------------
        397        32811
SQL>
```

Next, you issue the following command, using the appropriate file and block numbers, to get a dump of the block you need:

```
SQL> ALTER SYSTEM DUMP DATAFILE 397 BLOCK 32811;
System altered.
SQL>
```

The preceding command will produce a block dump in the default trace directory (UDUMP) of the Oracle database. Listing 4-2 shows part of the output of this command.

Listing 4-2. *A Sample Block Dump*

```
Dump file /a03/app/oracle/admin/pasu/udump/pasu_ora_29673.trc
Oracle Database 10g Enterprise Edition Release 10.1.0.2.0 - 64bit Production
With the Partitioning, OLAP and Data Mining options
*** 2005-05-01 10:59:05.905
*** ACTION NAME:() 2005-05-01 10:59:05.880
*** MODULE NAME:(SQL*Plus) 2005-05-01 10:59:05.880
*** SERVICE NAME:(SYS$USERS) 2005-05-01 10:59:05.880
*** SESSION ID:(207.10866) 2005-05-01 10:59:05.880
Start dump data blocks tsn: 110 file#: 397 minblk 32811 maxblk 32811
buffer tsn: 110 rdba: 0x6340802b (397/32811)
scn: 0x0001.610ac43d seq: 0x01 flg: 0x04 tail: 0xc43d2301
```

```
frmt: 0x02 chkval: 0x882e type: 0x23=PAGETABLE SEGMENT HEADER
  Extent Control Header
  -----------------------------------------------------------------
Extent Header:: spare1: 0  spare2: 0  #extents: 59  #blocks: 483328
                 last map  0x00000000  #maps: 0      offset: 2720
Highwater::  0x63826009  ext#: 58     blk#: 8192   ext size: 8192
#blocks in seg. hdr's freelists: 0
  #blocks below: 479093
  mapblk  0x00000000  offset: 58
                  Unlocked
  --------------------------------------------------------
Low HighWater Mark :
Highwater:: 0x6381ef7e  ext#: 4   blk#: 3957 ext size: 8192
  #blocks in seg. hdr's freelists: 0
  #blocks below: 36725
  mapblk  0x00000000  offset: 4
  Level 1 BMB for High HWM block: 0x63824028
  Level 1 BMB for Low HWM block: 0x6381e018
  --------------------------------------------------------
  Segment Type: 1 nl2: 0      blksz: 8192   fbsz: 0
  L2 Array start offset:  0x00001438
  First Level 3 BMB:  0x6340802a
  L2 Hint for inserts:  0x63408029
  Last Level 1 BMB:  0x63824028
  Last Level II BMB:  0x63412029
  Last Level III BMB:  0x6341202a
Map Header:: next 0x00000000 #extents:59 obj#:4916681 flag: 0x10000000
. . .
End dump data blocks tsn: 110 file#: 397 minblk 32811 maxblk 32811
```

It is possible to interpret and read dump data to find details about the information in a table or index. Let's look at a simple example that shows how you can get the table name from the preceding block dump information. Take the `obj#` shown in second-to-last line, and run the following query:

```
SQL> SELECT name
   2 FROM sys.obj$
  3* WHERE  obj#='4916681';
NAME
---------------
PERSONS
SQL>
```

The previous example is trivial, but it demonstrates how you can derive information straight from a database block dump. Of course, if you need more significant data from the dumps, you'd have to employ more rigorous techniques.

Extents

When several contiguous data blocks are combined, they are called an *extent.* When you create a database object like a table or index, you allocate it an initial amount of space, called the *initial extent,* and you also specify the size of the next and subsequent extents and the maximum number of extents for that object. Once allocated to a table or index, the extents remain allocated to that particular object, unless you drop the object from the database, in which case the space will revert to the pool of allocatable free space in the database.

Segments

A set of extents forms the next higher unit of data storage, the *segment*. Oracle calls all the space allocated to any particular database object a segment. So if you have a table called Customer, you simply refer to the space allocated to it as the "Customer segment." When you create an index, it will have its own segment named after the index name. Data and index segments are the most common type of Oracle segments. There are also temporary segments and rollback segments.

Tablespaces

Oracle databases are logically divided into one or more *tablespaces*. An Oracle tablespace is a logical entity that contains the physical data files. Tablespaces store all the usable data of the database, and the data in the tablespaces is physically stored in one or more data files. Data files are Oracle-formatted operating system files.

The tablespace is a purely logical construct and is the primary logical storage structure of an Oracle database. You usually should keep related tables together in the same tablespace, since the tablespace also acts as the logical container for logical segments such as tables.

How big you make your tablespaces depends on the size of your tables and indexes and the total amount of data in the database—there are no rules about the minimum or maximum size of tablespaces (the maximum size is too large to be of any practical consequence). It is quite common to have tablespaces that are 100GB in size coexisting in the same database with tablespaces as small as 1GB or even much smaller.

The data files that contain the data for the tablespaces in a database together constitute the total amount of physical space assigned to a particular database. (The size of a tablespace is the sum of the sizes of the data files that contain its data, and if you add up the sizes of the tablespaces or the sizes of all the data files, you will get the size of the database itself.) If you're running out of space in your database because you're adding new data, you need to create more tablespaces with new data files, add new data files to existing tablespaces, or make the existing data files of a tablespace larger. You'll learn how to perform each of these tasks in Chapter 5.

There is no hard and fast rule regarding the number of tablespaces you can have in an Oracle database. The following five tablespaces are generally the default tablespaces that all databases must have, even though it's possible to create and use a database with just the first two:

- System tablespace
- Sysaux tablespace
- Undo tablespace
- Temporary tablespace
- Default permanent tablespace

Traditionally, Oracle DBAs have used dozens and sometimes even hundreds of tablespaces to store all their application tables and indexes, and if you really think you need a large number of tablespaces to group all related application tables and indexes together, that's okay. However, you aren't required to use a large number of tablespaces. Today, most organizations use logical volume managers (which were discussed in Chapter 3) to stripe the logical volumes and the data files over a number of physical disks. Thus, a large tablespace could span several physical disks. Previously, it was necessary to create tablespaces on different physical disks to avoid I/O contention, but with today's disk organization structures you don't have that problem, and you can make do with fewer tablespaces if you wish. You can use just one tablespace for all your application data if you wish, since the data files that are part of the tablespace are going to be spread out over several disks anyway. This is also why the traditional requirement to separate tables and index data in different tablespaces isn't really valid anymore.

WHY TABLESPACES?

Tablespaces perform a number of key functions in an Oracle database, but the concept of a tablespace is not common to all relational databases. For instance, the Microsoft SQL Server database doesn't use this concept at all. Here's a brief list of the benefits of using tablespaces:

- Tablespaces make it easier to allocate space quotas to various users in the database.
- Tablespaces enable you to perform partial backups and recoveries based on the tablespace as a unit.
- Because a large object like a data warehouse partitioned table can be spread over several tablespaces, you can increase performance by spanning the tablespace over several disks and controllers.
- You can take a tablespace offline without having to bring down the entire database.
- Tablespaces are an easy way to allocate database space.
- You can import or export specific application data by using the import and export utilities at the tablespace level.

Tablespaces are now used mainly to separate related groups of tables and indexes. This may be important for you if you need to transport tablespaces across different databases and platforms using the Oracle Data Pump utility, or if you use different database block sizes for different tablespaces. If you don't think you'll be performing these administrative tasks using tablespaces, you can conceivably use just a couple of tablespaces to store all the data in your database.

Block Sizes and Tablespaces

Each tablespace uses the default block size for the database, unless you create a tablespace with a different nonstandard block size. As you've already seen, Oracle lets you have multiple block sizes in addition to the default block size. Because tablespaces ultimately consist of Oracle data blocks, this means that you can have tablespaces with different Oracle block sizes in the same database. This is a great feature, and it gives you the opportunity to pick the right block size for a tablespace based on the data structure of the tables it contains.

The customization of the block size for a tablespace provides several benefits:

- *Optimal disk I/O*: Remember that the Oracle server has to read the table data from mechanical disks into the buffer cache area for processing. One of your primary goals as a DBA is to optimize the expensive I/O involved in reading from and writing to disk. If you have tables with very long rows, you are better off with a larger block size—each read will fetch more data than you'd get with a smaller block size, and you'll need fewer read operations to get the same amount of data. Tables with large object (LOB) data will also benefit from a very large block size. On the other hand, tables with small row lengths can use a small block size as the building block for the tablespace. If you have large indexes in your database, you will need a large block size for their tablespace, so that each read will fetch a larger number of index pointers.
- *Optimal caching of data*: Oracle provides separate pools for the various block sizes, and this leads to a better use of Oracle's memory. I discuss this in the following sections.
- *Easier transport of tablespaces*: If you have tablespaces with multiple block sizes, it's easier to use Oracle's "transport tablespaces" feature. In Chapter 13, you'll find examples showing you how to transport tablespaces between databases.

■ **Note** Each Oracle tablespace consists of one or more operating system data files, and a data file can only belong to one tablespace. At database creation time, the only two tablespaces you *must* have are the System tablespace (the key Oracle tablespace, which contains Oracle's data dictionary), and the Sysaux tablespace (which is auxiliary to the System tablespace and contains data used by various Oracle products and features). Oracle will automatically create the System tablespace first, followed by the Sysaux tablespace, but you provide a data file for each. Later on, you can add and drop tablespaces as you wish, but you can't drop or rename the System and Sysaux tablespaces.

Temporary Tablespaces

Users need a temporary location to perform certain activities, such as sorting, and if you don't provide a designated *temporary tablespace* for them, they end up using the System tablespace. Given the importance of the System tablespace, which contains the data dictionary tables, along with other important information, it is obvious why you *must* have a temporary tablespace. Oracle allows you to create this temporary tablespace at database creation time, and all users will automatically use this as their default temporary tablespace. In addition, if you choose the Oracle-recommended Automatic Undo Management over the manual rollback-segment management mode, you'll also need to create an *undo tablespace* at database creation time.

Thus, although only the System and Sysaux tablespaces are absolutely mandatory, your database should also have a temporary and an undo tablespace when you initially create it. The System, Sysaux, temporary, and undo tablespaces all help manage Oracle system activity. The only tablespaces that Oracle creates automatically are the System and Sysaux tablespaces, though. Oracle creates these two tablespaces as part of the new database creation process.

Of course, you must ultimately also create separate application tablespaces to store your data and indexes. It is these tablespaces that will constitute the bulk of the total database size.

Dictionary-Managed vs. Locally Managed Tablespaces

You have a choice between two kinds of tablespaces: *dictionary-managed* tablespaces and *locally managed* tablespaces, which differ in how Oracle allocates extents in the tablespace.

In the case of dictionary-managed tablespaces, every time a table or other object needs to grow, Oracle checks its data dictionary to ensure that there's free disk space to allocate to the object, and then updates its free-space information after allocating a new extent to the object. Therefore, when you execute a SQL statement that inserts a large number of rows, for example, Oracle may well execute some additional SQL in the background in order to allocate more space to the table you are inserting data into. (SQL operations that occur when you consult the data dictionary are referred to as *recursive* SQL.) In addition to activity taking place in the data dictionary when additional extents are required, there's also activity in the undo segments, since the update activity in the data dictionary tables needs to be recorded in those segments. This extra activity when an object is trying to grow could occasionally lead to a performance slowdown.

Locally managed tablespaces keep the space-management information in the data files themselves, and the tablespaces automatically track the free or used status of blocks in each data file. The information about the free and used space in the data files is kept in bitmaps within the data file headers—*bitmaps* are maps that use bits to keep track of the space in a block or a group of blocks. Remember that when an object needs to grow, Oracle will assign new space in units of extents, not in terms of individual data blocks. So when a new extent needs to be allocated to an object, Oracle will select the first free data file and look up its bitmap to see if it has enough free contiguous data blocks. If so, Oracle will allocate the extent and then change the bitmap in that data file to show the new used status of the blocks in the extent.

During this process, the data dictionary isn't used in any way, so recursive SQL operations are significantly reduced. Rollback information is not generated during this updating of the bitmaps in the data files. Thus, the use of bitmaps in locally managed tablespaces leads to performance gains when compared to dictionary-managed tablespaces.

■ **Tip** Locally managed tablespaces are the default in Oracle Database 10*g*.

Create locally managed tablespaces to take advantage of their superior space-management abilities. The benefits are especially significant if your database is an OLTP database with numerous inserts, deletes, and updates taking place on a continuous basis.

Commonly Used Tablespaces

Besides the System and Sysaux tablespaces, you'll most likely also have undo and temporary tablespaces. You'll also use several other "permanent" tablespaces to hold your data and indexes.

Here's a summary of the key types of tablespaces you're likely to encounter:

- Bigfile tablespaces are tablespaces with a single large data file, whose size can range from 8 to 128 terabytes, depending on the database block size. Thus, your database could conceivably be stored in just one bigfile tablespace.
- Smallfile tablespaces can contain multiple data files, but the files cannot be as large as a bigfile data file. Smallfile tablespaces, which are the traditional tablespaces, are the default in Oracle Database 10*g*, and Oracle creates both System and Sysaux tablespaces as smallfile tablespaces.
- Temporary tablespaces contain data that persists only for the duration of a user's session. Usually Oracle uses these tablespaces for sorting and similar activities for users.
- Permanent tablespaces include all the tablespaces that aren't designated as temporary tablespaces.
- Undo tablespaces contain undo records, which Oracle uses to roll back, or undo, changes to the database.
- Read-only tablespaces don't allow write operations on the data files in the tablespace. You can convert any normal (read/write) tablespace to a read-only tablespace in order to protect data or to eliminate the need to perform backup and recovery of large data files that don't change.

Physical Database Structures

The Oracle database consists of the following three main types of files:

- *Data files*: These files store the table and index data.
- *Control files*: These files record changes to all database structures.
- *Redo log files*: These online files contain the changes made to table data.

In addition to these three types of files, an Oracle database makes use of several other operating system files to manage its operations. These include initialization files (like init.ora and the SPFILE), network administration files (like tnsnames.ora and listener.ora), alert log files, trace files, and the password file. Although these are referred to as *physical files*, to distinguish them from the

logical entities they contain, understand that from an operating system point of view, even these files are not really physical, but rather are logical components of the actual physical disks used by the operating system.

Oracle Data Files

Oracle data files make up the largest part of the physical storage of your database. A data file can belong to only one database, and one or more data files constitute the logical entity called the tablespace, which I described earlier in this chapter. Oracle data files constitute most of a database's total space.

When the database instance needs to read table or index data, it reads that from the data files on disk, unless that data is already cached in Oracle's memory. Similarly, any new table or index data or updates to existing data will be written to the data files for permanent storage.

The Control File

The *control file* is a file that the Oracle DBMS maintains to manage the state of the database, and it is probably the single most important file in the Oracle database. Every database has one control file, but due to the file's importance, multiple identical copies (usually three) are maintained—when the database writes to the control file, all copies of the file get written to. The control file is critical to the functioning of the database, and recovery is difficult without access to an up-to-date control file. Oracle creates the control file (and the copies) during the initial database creation process.

The control file contains the names and locations of the data files, redo log files, current log sequence numbers, backup set details, and the all-important *system change number* (SCN), which indicates the most recent version of committed changes in the database—information that is not accessible by users even for reading purposes. Only Oracle can write information to the control file, and the Oracle server process continually updates the control file during the operation of the database.

Control files are vital when the Oracle instance is operating. When you turn the instance on, Oracle reads the control file for the location of the data and log files. During the normal operation of the database, the control file is consulted periodically for necessary information regarding virtually every structure of the database.

The control file is also important in verifying the integrity of the database and when recovering the database. The checkpoint process instructs the database writer to write data to the disk when some specific conditions are met, and the control file notes all checkpoint information from the online redo log files. This information comes in handy during a recovery—the checkpoint information in the control file enables Oracle to decide how far back it needs to go in recovering data from the online redo log files. The checkpoint indicates the SCN up to which the data files are already written to the data files, so the recovery process will disregard all the information in the online redo log files before the checkpoint noted in the control file.

When you start an Oracle instance, the control file is consulted first, to identify all the data files and the redo log files that must be opened for database operations.

■ **Note** The checkpoint process is discussed in more detail in the "The Checkpoint" section later in this chapter.

Due to its obvious importance, Oracle recommends that you keep multiple copies of the control file.

The Redo Log Files

The Oracle *redo log files* record all the changes made to the database, and they are vital during the recovery of a database. If you need to restore your database from a backup, you can recover the latest changes made to the database from the redo log files. The set of redo log files that are currently being used to record the changes to the database are called *online redo log files*. These logs can be archived or copied to a different location before being reused, and the saved logs are called *archived redo logs*.

Oracle writes all final changes made to data (committed data) first to the redo log files, before applying those changes to the actual data files themselves. Thus, if a system failure prevents these data changes from being written to the permanent data files, Oracle will use the redo logs to recover all transactions that committed but couldn't be applied to the data files. Thus, redo log files guarantee that no committed data is never lost. If you have all the archived redo logs since the last database backup, and a set of the current redo logs as well, you can always bring a database up to date.

■ **Note** Current redo log files are often referred to as *online redo logs* to distinguish them from the older saved or archived redo log files.

Redo log files consist of *redo records,* which are groups of *change vectors,* each referring to a specific change made to a data block in the Oracle database. A single transaction may involve multiple changes to data blocks, so it may have more than one redo record. Initially, the contents of the log are kept in the *redo log buffer* (a memory area), but they are transferred to disk very quickly. If your database comes down without warning, the redo log can help you determine whether all transactions were committed before the crash or if some were still incomplete.

Oracle redo log files contain the following information about database changes made by transactions:

- Indicators specifying when the transaction started
- The name of the transaction
- The name of the data object that was being updated (e.g., an application table)
- The "before image" of the transaction (the data as it was before the changes were made)
- The "after image" of the transaction (the data as it was after the transaction made the changes)
- Commit indicators that indicate whether and when the transaction completed

When a database crashes, all transactions, both uncommitted as well as committed, have to be applied to the data files on disk, using the information in the redo log files. All redo log transactions that have both a begin and a commit entry must be redone, and all transactions that have a begin entry but no commit entry must be undone. (Redoing a transaction in this context simply means that you apply the information in the redo log files to the database; you do not rerun the transaction itself.) Committed transactions are thus re-created by applying the "after image" records in the redo log files to the database, and incomplete transactions are undone by using the "before image" records in the undo tablespace. Redo log files are an essential part of database management, and they are one of the main ways you enforce database consistency.

Oracle requires that every database have at least two redo log groups, each group consisting of at least one individual log file member. Oracle writes to one redo log file until it gets to the end of the redo log file, at which point it performs a log switch and starts writing to the second log file (and then to the third, if it exists).

By default, Oracle will write over the contents of a redo log file, unless you choose to archive your redo files. Oracle recommends that you archive the filled-up redo log files, so you can maintain a complete record of all the changes made to the database since the last backup. If you archive your redo log files, you are said to be running your database in the *archivelog* mode. Otherwise, you're running in *noarchivelog* mode.

Because of the critical importance of the redo log files in helping recover from database crashes, Oracle recommends *multiplexing* (maintaining multiple copies of) the redo log files. Multiplexing the online redo log files by placing two or more copies of the redo logs on different disk drives will ensure that you won't easily lose data changes that haven't been recorded in your data files.

The SPFILE

When you create a new database, you specify the initialization parameters for the Oracle instance in a special configuration file called the *server parameter file* (SPFILE). You can also use an older version of the configuration file called the init.ora file, but Oracle recommends the use of the more sophisticated SPFILE. In the SPFILE, you specify the memory limits for the instance, the locations of the control files, whether and where the archived logs are saved, and other settings that determine the behavior of the Oracle database server. You can't, however, edit the SPFILE manually, as you could the init.ora file, since the SPFILE is a binary file.

The SPFILE is always stored on the database server, thus preventing the proliferation of parameter files that sometimes occurs with the use of the init.ora file. By default, the SPFILE (and the init.ora file) is placed in the ORACLE_HOME/dbs directory in UNIX systems and the ORACLE_HOME\database directory in Windows systems. The ORACLE_HOME directory is the standard location for the Oracle executables.

■ **Note** You'll find a detailed discussion of the SPFILE, including how to create one from your init.ora file, in Chapter 9, where you will learn about creating Oracle databases.

Oracle allows you to change a number of the initialization parameters after you start up the instance; these are called *dynamic initialization parameters*. Unlike the traditional init.ora initialization file, the SPFILE can automatically and dynamically record the new values of dynamic parameters after you change them, ensuring that you don't forget to incorporate the changes. The rest of the parameters can't be changed dynamically, and you'll have to restart your instance if you need to modify any of those parameters.

You can use the V$SPPARAMETER data dictionary view to look at the initialization parameter values you have explicitly set in the SPFILE for your database. (The analogous view, if you are using the init.ora file, is the V$PARAMETER view.) In addition to the parameter values you set explicitly in the SPFILE, the V$SPPARAMETER view shows all the default values for all database configuration parameters (the values in effect in the instance right now).

Chapter 9 has a more complete discussion of the SPFILE.

■ **Caution** Sometimes you'll see references to undocumented or hidden Oracle parameters. These parameters usually have an underscore (_) prefix. Don't use them unless you're requested to do so by Oracle support experts or other trustworthy sources.

The Password File

The *password* file is an optional file in which you can specify the names of database users who have been granted the special SYSDBA or SYSOPER administrative privileges, which enable them to perform privileged operations, such as starting, stopping, backing up, and recovering databases. Chapter 10 shows you how to create and maintain the password file.

The Alert Log File

Every Oracle database has an *alert log* named alert*db_name*.log (where *db_name* is the name of the database). The alert log captures major changes and events that occur during the running of the Oracle instance, including log switches, any Oracle-related errors, warnings, and other messages. In addition, every time you start up the Oracle instance, Oracle will list all your initialization parameters in the alert log, along with the complete sequence of the start-up process. You can also use the alert log to automatically keep track of tablespaces that are created and data files that are added or resized.

The alert log can come in handy during troubleshooting—it is usually the first place you should check to get an idea about what was happening inside the database when a problem occurred. In fact, Oracle support may ask you for a copy of the pertinent sections of the alert log during their analysis of database problems.

Oracle puts the alert log in the location specified for the `BACKGROUND_DUMP_DEST` initialization parameter. If you don't specify a value for this parameter, Oracle places the alert log in a default location. For example, on HP-UX machines, the default location for the alert log is $ORACLE_HOME/rdbms/log. Commonly, it is located in a directory called *bdump*, which stands for background dump directory. To find out where the alert log is located, issue the following command:

```
SQL> SHOW PARAMETER background dump
NAME                    TYPE      VALUE
------------------------------------------
background_core_dump    string    partial
background_dump_dest    string    /u01/app/oracle/product/10.2.0/db_1/orcl/bdump
```

To see if there are any Oracle-related errors in your alert log, simply issue the following command (finance is the database name in this example):

```
$ grep ORA- alert_finance.log
ORA-1503 signalled during: CREATE CONTROLFILE SET DATABASE "FINANCE" RESETLOGS...
ORA-1109 signalled during: ALTER DATABASE CLOSE NORMAL...
ORA-00600: internal error code, arguments:[12333], [0], [0], [0], [], [], [], []
```

As you can see, several Oracle errors are listed in the alert log for the database finance. A regular scan of your database for all kinds of Oracle errors should be one of your daily database management tasks. You can easily schedule a script to scan the alert log and then e-mail you the results. You can also use the OEM Database Control (or Grid Control) interface to quickly review any errors in your alert log files.

Trace Files

Oracle requires that you specify three different trace file directories in your initialization file: the background dump directory, the core dump directory, and the user dump directory.

You specify the background dump directory using the `BACKGROUND_DUMP_DEST` parameter. This directory holds the debugging trace files for the background processes (LGWR, DBWn, and so on) that Oracle writes during instance operation. The background dump directory also contains the alert log file for the database instance (discussed in the previous section).

You specify the location of the core dump directory with the CORE_DUMP_DEST parameter. The core dump directory holds any core files generated during major errors such as the ORA-600 internal Oracle software errors.

You specify the location of the user dump directory using the USER_DUMP_DEST initialization parameter. The Oracle server will write all debugging trace files on behalf of a user process to the user dump directory. All trace files you generate using Oracle's SQL tracing features (explained in Chapters 21 and 22) will show up here.

Data Files and Tablespaces

To be able to use the disk for storing your data, directories and a file system must be created for you by the system administrator. You also need all the proper rights to read from and write to these directories and files. Then, when you create a tablespace, you assign it these data files. Before you create a database, your system administrator will assign a certain amount of disk space for the database based on your initial sizing estimates. All the administrator gives you are the assigned mount points for the various disks (for example, /prod01, /prod02, /prod03, and so on). You then need to create your directory structure under the mount points. After you install your software and create the Oracle administrative directories, you can use the remaining file system space for storing database objects, such as tables and indexes.

Oracle-managed files, which were introduced in Oracle8*i* (and which we'll discuss shortly), simplify the administration of Oracle databases. The Oracle Managed Files (OMF) feature eliminates the need for you to manage operating system files. You simply specify your database operations in terms of database objects, without using filenames.

For example, suppose you create a tablespace called customer01 with a 500MB data file. As you load more data into your database, Oracle will allocate new extents to the database tables by allocating space from the data file. When the table uses up almost all of the initial 500MB space allocation, you need to enlarge the tablespace by adding a new data file to it. You may alternatively increase the size of the existing data file by resizing it as well. If you don't, the table can't increase in size, and any attempts to add data to it will result in an error.

Although the data itself is placed in actual data files, there is no direct link between the tables and indexes and the data files they are placed in. These objects are only linked to the logical tablespace; it is the tablespace that is linked to the data files. Thus, Oracle maintains a separation between the logical objects (such as tables) and the physical data files. In other words, there is no direct connection during object creation or growth between the object and the data files it resides in. You can create or move an existing table or index by specifically declaring the tablespace, but you can't specify a data file directly.

Oracle Managed Files

The OMF feature aims at relieving DBAs of their traditional file-management tasks. When you use the OMF feature, you don't have to worry about the names and locations of the physical files. Instead, you can focus on the objects you're creating. Oracle will automatically create and delete files on the operating system as needed.

The OMF-based files are ideal for test and small databases, but if you have a terabyte-sized database with a large number of archived logs and redo logs, you need flexibility, which the OMF file system can't provide.

OMF drastically simplifies both the initial database creation as well as the management tasks. If you want to use OMF with your database, read the discussion of OMF in Chapter 18, where you'll learn how to create and manage OMF-based files.

Oracle Processes

Oracle server processes running under the operating system perform all the database operations, such as inserting and deleting data. These Oracle processes, together with the memory structures allocated to Oracle by the operating system, form the working Oracle instance. There is a set of mandatory Oracle processes that need to be up and running for the database to function at all. Other Oracle processes are necessary only if you are using certain specialized features of Oracle (such as replicated databases).

A *process* is essentially a connection or thread to the operating system that performs a task or job. The Oracle processes you'll encounter in this section are continuous, which means that they come up when the instance starts, and they stay up for the duration of the instance's life. Thus, they act like Oracle's hooks into the operating system's resources. A *process* on a UNIX system is analogous to a *thread* on a Windows system.

■ **Note** A "process" on a Windows Oracle installation is somewhat different from a "process" on a UNIX system. Please refer to the discussion on managing Oracle on Windows in Chapter 20 for a full explanation of Windows-based Oracle installations.

Oracle processes are divided into two general types both for efficiency and to keep client processes separate from the database server's tasks:

- *User processes*: These processes are responsible for running the application that connects the user to the database instance.
- *Oracle processes*: These processes perform the Oracle server's tasks, and you can divide them into two major categories: *server processes* and *background processes.* Together, these processes perform all the actual work of the database, from managing connections to writing to logs and data files to monitoring the user processes.

Interaction Between the User and Oracle Processes

User processes run application programs and Oracle tools, such as SQL*Plus. The user processes communicate with the server processes through the user interface and request that the Oracle server processes perform work on their behalf. Oracle responds by having its server processes service the user processes' requests. It's the job of the server processes to monitor user connections, accept requests for data, and return the results to the users. All `SELECT` requests, for example, involve reading data from the database, and it's the server processes that return the output of the `SELECT` statement back to the users.

You'll examine the two types of Oracle processes—the server processes and the background processes—in detail in the following sections.

The Server Process

When you run an Oracle tool, such as the OEM Database Control or the SQL*Plus interface, Oracle creates a user process for you. An Oracle session is defined as a specific connection of a user to the Oracle instance through the Oracle user process. The session duration lasts from the time you connect to the database by providing a username/password combination until you log out.

The *server process* is the process that services an individual user process. Each user connected to the database has a separate server process created for the duration of the session. The server process is created to service the user's process and is used by the user process to communicate with

the Oracle database server. When the user submits a request to select data, for example, the server process created for that user's application checks the syntax of the code and executes the SQL code. It then reads the data from the data files into the memory blocks. (If another user intends to read the same data, the second user's server process will read it not from disk again, but from Oracle's memory, where the data usually remains for a while.) Finally, the server process returns the requested data to the user.

The most common configuration for the server process is to assign each user a *dedicated* server process. However, Oracle provides for a more sophisticated means of servicing several users through the same server process, called the *shared server architecture,* which you'll learn about in more detail in Chapter 10.

Under the dedicated server process approach, each user has a one-to-one connection to the database through a dedicated server process. When you use the shared server architecture, several users connect through a *dispatcher* and use a shared server process. Even though the dedicated server approach is most commonly used, is easier to set up and tune, and is fine in most cases, it's better under some circumstances to use a shared server process, which helps conserve critical system resources, such as memory.

You can also configure shared server *connection pooling*. Connection pooling lets you reuse existing timed-out connections to service other active sessions. You can also configure shared server *session multiplexing,* which combines multiple sessions for transmission over the same network connection.

The Background Processes

The *background processes* are the real workhorses of the Oracle instance—they enable large numbers of users to concurrently and efficiently use information stored in database files. Oracle creates these processes automatically when you start an instance, and by being continuously hooked into the operating system, these processes relieve the Oracle software from having to repeatedly start numerous, separate processes for the various tasks that need to be done on the operating system's server. Each of the Oracle background processes is in charge of a separate task, thus increasing the efficiency of the database instance. These processes are automatically created by Oracle when you start the database instance, and they terminate when the database is shut down.

Table 4-1 lists the mandatory background processes that run in all Oracle databases. There are other specialized background processes that you'll need to use only if you're implementing certain advanced Oracle features.

Table 4-1. *Key Oracle Background Processes*

Background Process	Function
Database writer	Writes modified data from the buffer cache to disk (data files)
Log writer	Writes redo log buffer contents to the online redo log files
Checkpoint	Updates the headers of all data files to record the checkpoint details
Process monitor	Cleans up after finished and failed processes
System monitor	Performs crash recovery and coalesces extents
Archiver	Archives filled online redo log files
Manageability Monitor	Performs database-manageability-related tasks
Manageability Monitor Light	Performs tasks like capturing session history and metrics
Memory manager	Coordinates the sizing of the SGA components
Job queue coordination process	Coordinates job queues to expedite job processes

I briefly discuss the main Oracle background processes in the following sections.

The Database Writer

Oracle doesn't modify data directly on the disks—all modifications of data take place in Oracle memory. The *database writer* (DBW*n*) process is then responsible for writing the "dirty" (modified) data from the memory areas known as *database buffers* to the actual data files on disk.

It is the database writer process's job to monitor the use of the database buffer cache, and if the free space in the database buffers is getting low, the database writer process makes room available by writing some of the data in the buffers to the disk files. The database writer process uses the *least recently used* (LRU) algorithm (or a modified version of it), which retains data in the memory buffers based on how long it has been since someone asked for that data. If a piece of data has been requested very recently, it's more likely to be retained in the memory buffers.

The database writer process writes dirty buffers to disk under the following conditions:

1. When the database issues a checkpoint
2. When a server process can't find a clean reusable buffer after checking a threshold number of buffers
3. Every 3 seconds

■ **Note** Just because a user commits a transaction, it is not made permanent by the database writer process with an immediate write to the database files. Oracle conserves physical I/O by waiting to perform a more efficient write of batches of committed transactions at once.

For very large databases or for databases performing intensive operations, a single database writer process may be inadequate to perform all the writing to the database files. Oracle provides for the use of multiple database writer processes to share heavy data modification workloads. You can have a maximum of 20 database writer processes (DBW0 through DBW9, and DBWa through DBWj). Oracle recommends using multiple database writer processes, provided you have multiple processors.

You can specify the additional database writer processes by using the `DB_WRITER_PROCESSES` initialization parameter in the SPFILE Oracle configuration file. If you don't specify this parameter, Oracle allocates the number of database writer processes based on the number of CPUs and processor groups on your server. For example, on my 32-processor HP-UX server, the default is four database writers (one database writer per eight processors), and in another 16-processor server, the default is two database writers.

Oracle further recommends that you first ensure that your system is using asynchronous I/O before deploying additional database writer processes beyond the default number—you may not need multiple database writer processes if so. (Even when a system is capable of asynchronous I/O, that feature may not be enabled.) If your database writer can't keep up with the amount of work even after asynchronous I/O is enabled, you should consider increasing the number of database writers.

The Log Writer

The job of the *log writer* (LGWR) process is to transfer the contents of the redo log buffer to disk. Whenever you make a change to a database table (whether an insertion, update, or deletion), Oracle writes the committed and uncommitted changes to a redo log buffer (memory buffer). The log writer process then transfers these changes from the redo log buffer to the redo log files on disk.

The log writer writes a commit record to the redo log buffer and writes it to the redo log on disk immediately, whenever a user commits a transaction.

The log writer writes all redo log buffer entries to the redo logs under the following circumstances:

- Every 3 seconds.
- When the redo log buffer is one-third full.
- When the database writer signals that redo records need to be written to disk. Under Oracle's write-ahead protocol, all redo records associated with changes in the block buffers must be written to disk (that is, to the redo log files on disk) before the data files on disk can be modified. While writing dirty buffers from the buffer cache to the storage disks, if the database writer discovers that certain redo information has not been written to the redo log files, it signals the log writer to first write that information, so it can write its own data to disk.

The redo log files, as you learned earlier, are vital during the recovery of an Oracle database from a lost or damaged disk.

The Checkpoint

The *checkpoint* (CKPT) process is charged with telling the database writer process when to write the dirty data in the memory buffers to disk. After telling the database writer process to write the changed data, the checkpoint process updates the data file headers and the control file to indicate when the checkpoint was performed. The purpose of the checkpoint process is to synchronize the buffer cache information with the information on the database disks.

Each checkpoint record consists of a list of all active transactions and the address of the most recent log record for those transactions. A checkpointing process involves the following steps:

1. Flushing the contents of the redo log buffers to the redo log files
2. Writing a checkpoint record to the redo log file
3. Flushing the contents of the database buffer cache to disk
4. Updating the data file headers and the control files after the checkpoint completes

There is a close connection between how often Oracle checkpoints and the recovery time after a database crash. Because database writer processes write all modified blocks to disk at checkpoints, the more frequent the checkpoints, the less data will need to be recovered when the instance crashes. However, checkpointing involves an overhead cost. Oracle lets you configure the database for *automatic checkpoint tuning,* whereby the database server tries to write out the dirty buffers in the most efficient way possible, with the least amount of adverse impact on throughput and performance. If you use automatic checkpoint tuning, you don't have to set any checkpoint-related parameters.

The Process Monitor

When user processes fail, the *process monitor* (PMON) process cleans up after them, ensuring that the database frees up the resources that the dead processes were using. For example, when a user process dies while holding certain table locks, the PMON process releases those locks so other users can use the tables without any interference from the dead process. In addition, the PMON process restarts failed server processes and dispatcher processes. The PMON process sleeps most of the time, waking up at regular intervals to see if it is needed. Other processes will also wake up the PMON process if necessary.

The PMON process automatically performs dynamic service registration. When you create a new database instance, the PMON process registers the instance information with the listener, which is the entity that manages requests for database connections (Chapter 10 discusses the listener in detail). This *dynamic service registration* eliminates the need to register the new service information in the listener.ora file, which is the configuration file for the listener.

The System Monitor

The *system monitor* (SMON) process, as its name indicates, performs system-monitoring tasks for the Oracle instance, such as these:

- Upon restarting an instance that crashed, SMON determines whether the database is consistent.
- SMON coalesces free extents if you use dictionary-managed tablespaces, which enables you to assign larger contiguous free areas on disk to your database objects.
- SMON cleans up unnecessary temporary segments.

Like the PMON process, the SMON process sleeps most of the time, waking up to see if it is needed. Other processes will also wake up the SMON process if they detect a need for it.

The File Mapping Monitor

File systems are increasingly complex, and to help you in monitoring I/O, Oracle provides the *file mapping monitor* (FMON) process to map files to immediate storage layers and physical devices. This will help you understand exactly how your data files are stored in a disk system managed by a Logical Volume Manager (LVM).

The FMON process interacts with mapping libraries provided by the operating system to perform the file mapping. The results are in the DBMS_STORAGE_MAP view.

The Archiver

The *archiver* (ARC*n*) process is used when the system is being operated in an *archivelog mode*—that is, the changes logged to the redo log files are being saved and not being overwritten by new changes. If you run your database in the *no archivelog mode*, Oracle will overwrite the redo log files with new redo log records. When you choose to run the instance in an archivelog mode, no such overwriting can take place—each filled log will be saved or archived in a special location.

The archiver process will archive the redo log files to the location you specify. You usually copy these archived logs to tape and send them to an offsite storage location to ensure you have a complete set of backups and archived redo logs so that you can perform a database recovery if the need arises.

If a huge number of changes are being made to your database, and your logs are consequently filling up very quickly, you can use multiple archiver processes up to a maximum of ten (ARC0 through ARC9). The `LOG_ARCHIVE_MAX_PROCESSES` parameter in the initialization file will determine how many archiver processes Oracle will start. If the log writer process is writing logs faster than the default single archiver process can archive them, the LGWR process automatically starts a new ARC*n* process, thus raising the number of processes from their default value of 1.

■ **Tip** If you aren't sure what new background processes are actually running in your database, just check the processes by issuing the `ps –eaf | grep ora` command in UNIX and Linux systems. For each active process, the process name and database name will be listed. For example, the log writer process will show up as ora_lgwr_pasprod, where pasprod is the name of the database. You can get a complete list of all the background processes (running and not running) by querying the V$BGPROCESS view.

The Manageability Monitor

The *manageability monitor* (MMON) process collects several types of statistics to help the database manage itself. For example, MMON collects the Automatic Workload Repository (AWR) snapshot information, which is the basis for the performance diagnostics capability of the Automatic Database Diagnostic Monitor (ADDM). MMON also issues alerts when database metrics violate their threshold values.

The Manageability Monitor Light

The *manageability monitor light* (MMNL) process shows up as the Manageability Monitor Process 2 when you query the V$BGPROCESS view. The process flushes data from the Active Session History (ASH) to disk whenever the buffer is full. The MMNL process also performs other manageability-related tasks, such as capturing session history data and computing database metrics.

The Memory Manager

The *memory manager* (MMAN) process coordinates the sizing of the memory components. MMAN keeps track of the sizes of the memory components and the pending resize operations. It observes the system and workload in order to determine the ideal distribution of memory, and it ensures that the needed memory is available.

The Job Queue Coordination Process

Oracle uses the *job queue coordination* (CJQO) process to schedule and run user jobs. The coordinator process dynamically spawns job queue slave processes (J000 through J999), which run the user jobs.

The Rebalance Master

The *rebalance master* (RBAL) process coordinates disk rebalancing activity when you use an Automatic Storage Management (ASM) storage system.

The ASM Rebalance

The *ASM rebalance* (ARB*n*) processes perform the disk rebalancing activity in an ASM instance.

The ASM Background

The *ASM background* (ASMB) process is present in all Oracle databases that use an ASM storage system. The ASMB process communicates with the ASM instance by logging into the ASM instance as a foreground process.

■ **Note** The RBAL and ORB*n* processes are used only if you use Oracle's Automatic Storage Management. When you use ASM, you must create an ASM instance, and that instance will use these processes to perform disk storage management. The OSMB process acts as the mediator between your database (when you're using ASM-based disk storage) and the ASM instance. I discuss ASM in detail in Chapter 17.

The Recovery Writer

There's a new type of log known as a *flashback log*, and it logs the before images of Oracle blocks from the new flashback buffers, which are located in the system global area (SGA), which is the name for Oracle's memory allocation. (I discuss the SGA in the "The System Global Area (SGA)" section, later in this chapter.) When you enable the new flashback database feature (which is explained in Chapter 16), Oracle starts the *recovery writer* (RVWR) process to write the flashback data from the flashback buffer to the flashback logs. In a sense, the RVWR'S job is analogous to that of the LGWR background process.

The Change Tracking Writer

Oracle tracks the physical location of database changes in a new file called the change-tracking file. Oracle's backup utility, the Recovery Manager (RMAN), uses the change-tracking file to determine which data blocks to read during an incremental backup, making the incremental backups faster by avoiding reading entire data files. The *change-tracking writer* (CTWR) process is the new Oracle background process that writes change information to the change-tracking file. You'll learn more about the CTWR process in Chapter 15, which discusses database backups.

Miscellaneous Background Processes

In addition to the background processes already described, there are other processes as well, such as the Queue Monitor Coordinator (QMNC), which spawns and coordinates queue slave processes, and the recoverer (RECO) process, which is used to coordinate distributed databases and other specialized processes.

■ **Note** Besides the processes discussed here, other Oracle background processes that perform specialized tasks may be running in your system. For example, if you use Oracle Real Application Clusters, you'll see a background process called the *lock* (LCK*n*) process, which is responsible for performing inter-instance locking.

Oracle Memory Structures

Oracle uses a part of its memory allocation to hold both program code and data, which makes processing much faster than if it had to fetch data from the disks constantly. These memory structures enable Oracle to share executable code among several users without having to go through all the preexecution processing every time a user invokes a piece of code.

The Oracle server doesn't always write changes to disk directly. It writes database changes to the memory area, and when it's convenient, it writes the changes to disk. Because accessing memory is many times faster than accessing physical disks (memory access is measured in nanoseconds, whereas disk access is measured in milliseconds), Oracle is able to overcome the I/O limitations of the disk system. The more your database performs its work in memory rather than in the physical disk storage system, the faster the response will be. Of course, as physical I/O decreases, CPU usage will also decrease, thus leading to a more efficient system.

THE HIGH COST OF DISK I/O

Although secondary storage (usually magnetic disks) is significantly larger than main memory, it's also significantly slower. A disk I/O involves either moving a data block from disk to memory (a disk read) or writing a data block to disk from memory (a disk write). Typically, it takes about 10–40 milliseconds (0.01–0.04 seconds) to perform a single disk I/O.

Suppose your update transaction involves 25 I/Os—you could spend up to 1 second just waiting to read or write data. In that same second, your CPUs could have performed millions of instructions—the update takes a negligible amount of time compared to the disk reads and disk writes. If you already have the necessary data in Oracle's memory, the retrieval time would be much faster, as memory read/writes take only a few nanoseconds. This is why avoiding or minimizing disk I/Os plays such a big role in providing high performance in Oracle databases.

Understanding Main Memory

All computers use memory, which actually consists of a hierarchy of different levels of memory. The heart of this hierarchy is *main memory*, which contains all the instruction executions and data manipulations. All main memories are random access memory (RAM), which means that you can read any byte in memory in the same amount of time. Typically, you can access main memory data in the 10–100 nanosecond range.

An important part of the information Oracle stores in the RAM allocated to it is the program code that is executing currently or that has been executed recently. If a new user process needs to use the same code, it's available in memory in a compiled form, making the processing time a whole lot faster. The memory areas also hold information about which users are locking a certain table, thereby helping different sessions communicate effectively. Most important, perhaps, the memory areas help in processing data that's stored in permanent disk storage. Oracle doesn't make changes directly to the data on disk: data is always read from the disks, held in memory, and changed there before being transferred back to disk.

It's common to use the term *buffers* to refer to units of memory. Memory buffers are page-sized areas of memory into which Oracle transfers the contents of the disk blocks. If the database wants to read (select) or update data, it copies the relevant blocks from disk to the memory buffers. After it makes any necessary changes, Oracle transfers the contents of the memory buffers to disk.

Oracle uses two kinds of memory structures, one shared and the other process-specific. The *system global area* (SGA) is the part of total memory that all server processes (including background processes) share. The process-specific part of the memory is known as the *program global area* (PGA), or *process-private memory*. The following sections examine these two components of Oracle's memory in more detail.

The System Global Area (SGA)

The SGA is the most important memory component in an Oracle instance. In large OLTP databases, especially, the SGA is a much larger and more important memory area than the PGA. In data warehousing environments, on the other hand, the PGA can be the more important Oracle memory area, because it critically influences the efficiency of large data sorts and hashes, which are commonly part of analytic computations in data warehouses.

The SGA's purpose is to speed up query performance and to enable a high amount of concurrent database activity. Because processing in memory is much faster than disk I/O, the size of the SGA is one of the more important configuration issues when you're tuning the database for optimal performance. When you start an instance in Oracle, the instance takes a certain amount of memory

from the operating system's RAM—the amount is based on the size of the SGA component in the initialization file. When the instance is shut down, the memory used by the SGA goes back to the host system.

The SGA isn't a homogeneous entity; rather, it's a combination of several memory structures. The following are the main components of the SGA:

- *Database buffer cache*: Holds copies of data blocks read from data files.
- *Shared pool*: Contains the library cache for storing SQL and PL/SQL parsed code in order to share it among users. It also contains the data dictionary cache, which holds key data dictionary information.
- *Redo log buffer*: Contains the information necessary to reconstruct changes made to the database by DML operations. This information is then recorded in the redo logs by the log writer.
- *Java pool*: Keeps the state of Java program execution.
- *Large pool*: Stores large memory allocations, such as RMAN backup buffers.
- *Streams pool*: Supports the Oracle Streams feature.

When you start the Oracle instance, Oracle allocates memory as needed until it reaches the size set in the `SGA_TARGET` initialization parameter, which sets the limit for the total memory allocation. If your total memory allocation is already at the `SGA_TARGET` limit, you can't dynamically increase memory to any SGA component without decreasing some other component's memory allocation. Oracle does allow you to exchange the memory from one dynamically sizable memory component to another.

For example, you can increase the memory assigned to the buffer cache by taking it from the shared pool. If you have certain jobs run only at specified times of the day, you can write a simple script that runs before the job executes and modifies the allocation of memory among the various components. After the job completes, you can have another script run that changes the memory allocation back to the original settings.

The next few sections discuss the various components of the SGA. You can manage the SGA yourself, by calibrating the memory you make available to the Oracle instance with the changing memory requirements of the running instance. However, the best way to manage the SGA is simply by using automatic shared memory management, which I introduce in the "Automatic Shared Memory Management" section, later in this chapter.

The Database Buffer Cache

The *database buffer cache* consists of the memory buffers that Oracle uses to hold the data read by the server process from data files on disk in response to user requests. Buffer cache access is, of course, much faster than reading the data from disk storage. When the users modify data, those changes are made in the database buffer cache as well. The buffer cache thus contains both the original blocks read from disk and the changed blocks that have to be written back to disk.

You can group the memory buffers in the database buffer cache into three components:

- *Free buffers*: These are buffers that do not contain any useful data, and, thus, the database can reuse them to hold new data it reads from disk.
- *Dirty buffers*: These contain data that was read from disk and then modified, but hasn't yet been written to the data files on disk.
- *Pinned buffers*: These are data buffers that are currently in active use by user sessions.

When a user process requests data, Oracle will first check whether the data is already available in the buffer cache. If it is, the server process will read the data from the SGA directly and send it to the user. If the data isn't found in the buffer cache, the server process will read the relevant data from the data files on disk and cache it in the database buffer cache. Of course, there must be free buffers available in the buffer cache for the data to be read into them. If the server process can't find a free buffer after searching through a threshold number of buffers, it asks the database writer process to write some of the dirty buffers to disk, thus freeing them up for writing the new data it wants to read into the buffer cache.

Oracle maintains a least recently used (LRU) list of all free, pinned, and dirty buffers in memory. It's the database writer process's job to write the dirty buffers back to disk to make sure there are free buffers available in the database buffer cache at all times. To determine which dirty blocks get written to disk, Oracle uses a modified LRU algorithm, which ensures that only the most recently accessed data is retained in the buffer cache. Writing data that isn't being currently requested to disk enhances the performance of the database.

The larger the buffer cache, the fewer the disk reads and writes needed and the better the performance of the database. Therefore, properly sizing the buffer cache is very important for the proper performance of your database. Of course, simply assigning an extremely large buffer cache can hurt performance, because you may end up taking more memory than necessary and causing paging and swapping on your server.

Using Multiple Database Buffer Cache Pools

Generally, a single default buffer cache is sufficient to serve the instance's memory needs. Assigning the same database buffer cache for all the database objects may not be very efficient at times, because different objects and various types of data may have different requirements as to how long they should be retained in the data cache. For example, table A may be accessed a hundred thousand times during a day, whereas table B may be accessed only twice during the same day. Clearly, it makes sense here to retain table A in the buffer cache throughout the day, so as to increase the speed of access, while table B can be removed after each use, to conserve space in the cache.

Oracle gives you flexibility in the use of the buffer cache by allowing you to configure the database buffer cache into multiple *buffer pools*. A buffer pool in this context is simply a part of the total buffer cache that is subject to different retention criteria for database objects like tables. For example, you can take a total buffer cache of 500MB and divide it into three pools, with 200MB in the first two pools and 100MB in the third. Once you have created separate buffer pools, you can assign a table exclusively to that buffer pool when you create that table. You can also use the `ALTER TABLE` or `ALTER INDEX` command to modify the type of buffer pool that a table or index should use. Table 4-2 lists the main types of buffer pools that you can configure.

Note that any database objects that you haven't assigned to the keep or the recycle buffer pool will be assigned to the default buffer pool, which is sized according to the value you provide for the `DB_CACHE_SIZE` initialization parameter. The keep and the recycle buffer pools are purely optional, while the default buffer pool is mandatory.

Remember that the main goal in assigning objects to multiple buffer pools is to minimize the misses in the data cache and thus minimize your disk I/O. In fact, all buffer caching strategies have this as their main goal. If you aren't sure which objects in your database belong to the different types of buffer caches, just let the database run for a while with some best-guess multiple cache sizes and query the data dictionary view `V$DB_CACHE_ADVICE` to get some advice from Oracle itself.

Table 4-2. *Main Buffer Pool Types*

Buffer Pool	Initialization Parameter	Description
Keep buffer pool	DB_KEEP_CACHE_SIZE	Keeps the data blocks always in memory. You may have small tables that are frequently accessed, so to prevent them from being aged out of the database buffer cache, you can assign the tables to the keep buffer cache when they are created.
Recycle buffer pool	DB_RECYCLE_CACHE_SIZE	Removes the data from the cache immediately after use. You need to use this buffer pool carefully, if you decide to use it at all. The recycle buffer pool will cycle out the object from the cache as soon as the transaction is over. Obviously, you would use the recycle buffer pool only for large tables that are infrequently accessed and that do not need to be retained in the buffer cache indefinitely.
Default buffer pool	DB_CACHE_SIZE	Contains all data and objects that are not assigned to the keep and recycle buffer pools.

Multiple Database Block Sizes and the Buffer Cache

As was mentioned earlier, you can have multiple block sizes for your database. You have to choose a standard block size first, and then you can choose up to four other nonstandard cache sizes.

The DB_BLOCK_SIZE parameter in your initialization parameter file determines the size of your *standard* block size in the database and frequently is the only block size for the entire database. The DB_CACHE_SIZE parameter in your initialization parameter file specifies the size (in bytes) of the cache of the standard block sized buffers. Notice that you don't set the *number* of database buffers; rather, you specify the *size* of the buffer cache itself in the DB_CACHE_SIZE parameter.

You can have up to five different database block sizes in your databases. That is, you can create your tablespaces with any one of the five allowable database block sizes. Although most databases use only a single standard block size (such as 4KB, 8KB, or 16KB), you can choose to use some or all of the four nonstandard block sizes as well. For example, you may have some data warehouse–type tables that will benefit from a high database block size, such as 32KB. However, most of the other tables in the database may serve online processing needs, and should use the standard block size of 4KB. If you happen to be using all four of the allowable nonstandard block sizes besides the standard block size buffers, you can create tablespaces with all five block sizes. However, before you can create these nonstandard block size tablespaces, you must configure nonstandard subcaches in the buffer caches for each nonstandard block size you wish to use. You can specify the nonstandard buffer cache subcaches by using the DB_*n*K_CACHE_SIZE initialization parameter, where *n* is the block size in kilobytes—it can take a value of 2, 4, 8, 16, or 32.

As you've seen, the database buffer cache can be divided into three pools: the default, keep, and recycle buffer pools. The total size of the buffer cache is the sum of memory blocks assigned to all the components of the database buffer cache. The keep and recycle buffer pools can only be created with the standard block size, but you can use up to five different nonstandard block sizes to configure the default buffer pool.

Here's an example that shows how you can specify different values for each of the buffer cache's subcaches in your initialization parameter file. In the example, the numbers on the right show the memory allocated to a particular type of buffer cache.

```
DB_KEEP_CACHE_SIZE  = 48MB
DB_RECYCLE_CACHE_SIZE = 24MB
DB_CACHE_SIZE = 128MB /* standard 4KB block size */
DB_2k_CACHE_SIZE =48MB /* 2KB non-standard block size */
DB_8k_CACHE_SIZE =192MB /* 8KB non-standard block size */
DB_16k_CACHE_SIZE = 384MB /* 16KB non-standard block size */
```

The total buffer cache size in this example will be the sum of all the above subcaches, which comes to about 824MB.

The Buffer Cache Hit Ratio

Buffer reads are much faster than reads from disk. The all-important principle in appropriately sizing the buffer cache is summarized in the phrase "touch as few blocks as possible," since disk I/Os necessary for reading data from Oracle blocks on disk are more time-consuming than reading the data from the SGA. This is why the buffer cache *hit ratio,* which measures the percentage of time users accessed the data they needed from the buffer cache (rather than requiring a disk read), is such an important indicator of performance of the Oracle instance.

You derive the buffer cache hit ratio as follows:

```
hit rate = (1 - (physical reads)/(logical reads)) * 100
```

In this calculation, the physical and logical reads (reads from disk and from memory, respectively) are accumulated from the start of the Oracle instance. So if you calculate the ratio on Monday morning after a restart on Sunday night, it will show a very low hit ratio. As the week progresses, the hit ratio could increase dramatically, because as more read requests come in, Oracle satisfies them with the data that is already in memory.

Unfortunately, Oracle does not give you any reliable rules or guidelines to indicate how much memory you should allocate for your buffer cache ratio or the SGA. Some trial and error with data loads should give you a good idea about the right size.

In Chapter 22, I present much more information on the proper tuning of the database buffer cache. A high buffer cache hit ratio doesn't always correlate with superior database performance. It is entirely possible for your database to have a very high hit ratio—say, in the high 90s—and still have a performance problem. For example, even if your total logical reads and hit ratio are high, your SQL queries could still be inefficient.

The Shared Pool

The shared pool is a very important part of the Oracle SGA, and sizing it appropriately for your instance will help avoid several types of Oracle instance bottlenecks. Unlike the database buffer cache, which holds actual data blocks, the shared pool holds executable PL/SQL code and SQL statements, as well as information regarding the data dictionary tables. The *data dictionary* is a set of key tables that Oracle maintains, and it contains crucial metadata about the database tables, users, privileges, and so forth.

Proper sizing of the shared pool area benefits you in a couple of ways. First, your response times will be better because you're reducing processing time—if you don't have to recompile the same Oracle code each time a user executes a query, you save time. Oracle will reuse the previously compiled code if it encounters the same code again. Second, more users can use the system because the reuse of code makes it possible for the database to serve more users with the same resources. Both the I/O rates and the CPU usage will diminish when your database uses its shared pool memory effectively.

The following sections discuss the library cache and the data dictionary cache, both of which are components of the shared pool.

The Library Cache

All application code, whether it is pure SQL code or code embedded in the form of PL/SQL program units, such as procedures and packages, is parsed first and executed later. Oracle stores all compiled SQL statements in the *library cache* component of the shared pool. The library cache component of the shared pool memory is shared by all users of the database. Each time you issue a SQL statement, Oracle first checks the library cache to see if there is an already parsed and ready-to-execute form of the statement in there. If there is, Oracle uses the library cache version, reducing the processing time considerably—this is called a *soft parse*.

If Oracle doesn't find an execution-ready version of the SQL code in the library cache, the executable has to be built fresh—this is called a *hard parse*. Oracle uses the library cache part of the shared pool memory for storing newly parsed code. If there isn't enough free memory in the shared pool, Oracle will jettison older code from the shared pool to make room for your new code.

All hard parses involve the use of critical system resources, such as processing power and internal Oracle structures, such as latches; you must make every attempt to reduce their occurrence. High hard-parse counts will lead to resource contention and a consequent slowdown of the database when responding to user requests.

You should make decisions about the library cache size based on hit and miss ratios on the library cache as discussed in Chapter 22. If your system is showing more than the normal amount of misses (meaning that code is being reparsed or re-executed often), it is time to increase the library cache memory. The way to do this is to increase the total memory allocated to the shared pool.

The Data Dictionary Cache

The *data dictionary cache* component of the shared pool primarily contains object definitions, usernames, roles, privileges, and other such information. When you run a segment of SQL code, Oracle first has to ascertain whether you have the privileges to perform the planned operation. It checks the data dictionary cache to see whether the pertinent information is there, and if not, Oracle has to read the information from the data dictionary into the data dictionary cache. Obviously, the more often you find the necessary information in the cache, the shorter the processing time. In general a data dictionary cache miss, which occurs when Oracle doesn't find the information it needs in the cache, tends to be more expensive than a library cache miss.

There is no direct way to adjust the data dictionary cache size. You can only increase or decrease the entire shared pool size. Therefore, the solution to a low data dictionary cache hit ratio or a low library cache hit ratio is the same: increase the shared pool size.

■ **Tip** A cache miss on either the data dictionary cache or the library cache component of the shared pool has more impact on database performance than a miss on the buffer pool cache. For example, a decrease in the data dictionary cache hit ratio from 99 percent to 89 percent leads to a much more substantial deterioration in performance than a similar drop in the buffer cache hit ratio.

The Redo Log Buffer

The *redo log buffer*, usually less than a couple of megabytes in size, and thus nowhere near the size of the database buffer cache and the shared pool cache, is nonetheless a crucial component of the SGA. When a server process changes data in the data buffer cache (via an insert, a delete, or an update), it generates redo data, which is recorded in the redo log buffer. The log writer process writes redo information from the redo log buffer in memory to the redo log files on disk.

You use the LOG_BUFFER initialization parameter to set the size of the redo log buffer, and it stays fixed for the duration of the instance. That is, you can't adjust the redo log buffer size dynamically, unlike the other components of the SGA.

The log writer process writes the contents of the redo log buffer to disk under any of the following circumstances:

- The redo log buffer is one-third full.
- Users commit a transaction.
- The database buffer cache is running low on free space and needs to write changed data to the redo log. The database writer instructs the log writer process to flush the log buffer's contents to disk to make room for the new data.

The redo log buffer is a circular buffer—the log writer process writes the redo entries from the redo log buffer to the redo log files, and server processes write new redo log entries over the entries that have been written to the redo log files. You only need to have a small redo log buffer, about 1MB or so. Large redo log buffers will reduce your log file I/O (especially if you have large or many transactions), but your commits will take longer as well.

The log writer process usually writes to the redo log files very quickly, even when its workload is quite heavy. You'll run into more problems if your redo log buffer size is too small than if it is too large. A redo log buffer that is too small will keep the log writer process excessively busy—it will be constantly writing to disk. Furthermore, if the log buffer is too small, it will frequently run out of space to accommodate new redo entries.

Oracle provides an option called *nologging* that lets you bypass the redo logs almost completely and thus avoid contention during certain operations (such as a large data load). You can also batch the commits in a long job, thus enabling the log writer process to more efficiently write the redo log entries.

The Large Pool and the Java Pool

The large pool is a purely optional memory pool, and Oracle manages it quite differently from the shared pool. Oracle uses the large pool mostly for accommodating Recovery Manager (RMAN) operations. You set the size of this pool in the initialization file by using the LARGE_POOL_SIZE parameter. The large pool memory component is important if you're using the shared server architecture.

The Java pool (set by using the JAVA_POOL_SIZE parameter) is designed for databases that contain a lot of Java code, so that the regular SGA doesn't have to be allocated to components using Java-based objects. Java pool memory is reserved for the Java Virtual Machine (JVM) and for your Java-based applications. The default size for this memory pool is 20MB, but if you're deploying Enterprise JavaBeans or using CORBA, you could potentially need a Java pool size greater than 1GB.

The Streams Pool

Oracle Streams is a technology for enabling data sharing among different databases and among different application environments. The Streams pool is the memory allocated to support Streams activity in your instance. If you manually set the Streams pool component by using the STREAMS_POOL_SIZE initialization parameter, memory for this pool is transferred from the buffer cache after the first use of Streams. If you use automatic shared memory management (discussed next), the memory for the Streams pool comes from the global SGA pool. The amount transferred is up to 10 percent of the shared pool size.

Automatic Shared Memory Management

In previous versions of Oracle, DBAs spent quite a bit of time pondering the sizing of the SGA. It wasn't uncommon for them to recalibrate the SGA size quite often as part of their instance-tuning efforts. In Oracle Database 10*g*, you can configure *automatic shared memory management* by using the new SGA_TARGET initialization parameter. All you need to do is assign a certain value for the SGA_TARGET parameter, and Oracle will automatically manage the distribution of this memory among the various components of the SGA. Oracle's allocation of the SGA memory to the various components isn't static, but changes with the changing workload of the database. Oracle can automatically manage the following five components of the SGA (the relevant Oracle initialization parameter is in parentheses):

- Database buffer cache (DB_CACHE_SIZE)
- Shared pool (SHARED_POOL_SIZE)
- Large pool (LARGE_POOL_SIZE)
- Java pool (JAVA_POOL_SIZE)
- Streams pool (STREAMS_POOL_SIZE)

As you can see, Oracle automatically tunes five components of the SGA, which are referred to as the automatically sized SGA parameters. You must still manage the rest of the SGA components yourself, even under automatic shared memory management. The following are the manually tunable components of the SGA:

- Keep buffer cache (DB_KEEP_CACHE_SIZE)
- Recycle buffer cache (DB_RECYCLE_CACHE_SIZE)
- Any nonstandard block size buffer caches (DB_*n*K_CACHE_SIZE)
- Redo log buffer (LOG_BUFFER)

Note that the first three components in this list are optional. As the DBA, you must set the value for each of the manual SGA components.

You can set up automatic shared memory management simply by setting the SGA_TARGET parameter to a positive value. Once you do this, Oracle will automatically tune the five auto-tuned SGA parameters, but not all of the SGA_TARGET's total size can be taken by the auto-tuned parameters—Oracle will first deduct the memory necessary for the manual SGA parameters from the SGA_TARGET size, and it will allocate the remainder for the auto-tuned parameters.

When you set the SGA_TARGET parameter to a positive value, the default value for the five auto-tuned SGA parameters will be zero, but if you set a specific value for any of the auto-tuned parameters, that value becomes the lower bound for that parameter. If there isn't enough memory left in the SGA to satisfy any values you select for the auto-tuned parameters, Oracle will just reduce the lower bound of those parameters to fit within the available memory.

The total size of the SGA will be the sum of the memory allocated to the auto-tuned SGA parameters, memory allocated to the manual SGA parameters, and fixed SGA and internal allocations.

■ **Note** If the SGA_TARGET parameter is set to zero (the default), the auto-tuned SGA parameters behave as in previous versions of Oracle.

You can learn more about automatic shared memory management in Chapter 22.

The Program Global Area (PGA)

Oracle creates a program global area (PGA) for each user when the user starts a session. This area holds data and control information for the dedicated server process that Oracle creates for each individual user. Unlike the SGA, the PGA is for the exclusive use of each server process and can't be shared by multiple processes. A session's logon information and persistent information, such as bind variable information and data type conversions, are still a part of the SGA, unless you're using a shared server configuration, but the runtime area used while SQL statements are executing is located in the PGA.

For example, a user's process may have some cursors (which are handles to memory areas where you store the values for variables) associated with it. Because these are the user's cursors, they are not automatically shared with other users, so the PGA is a good place to save those private values. Another major use of the PGA is for performing memory-intensive SQL operations that involve sorting, such as queries involving `ORDER BY` and `GROUP BY` clauses. These sort operations need a working area, and the PGA provides that memory area.

■ **Note** For most OLTP databases, where transactions are very short, the PGA use is quite low. On the other hand, complex, long-running queries, which are more typical of DSS environments, require larger amounts of PGA memory.

You can classify the PGA memory into the following types:

- Private SQL area: This area of memory holds SQL variable bind information and runtime memory structures. Each session that executes a SQL statement will have its own private SQL area.
- Runtime area: The runtime area is created for a user session when the session issues a `SELECT`, `INSERT`, `UPDATE`, or `DELETE` statement. After an `INSERT`, `DELETE`, or `UPDATE` statement is run, or after the output of a `SELECT` statement is fetched, the runtime area is freed by Oracle.

If a user's session uses complex joins or heavy sorting (grouping and ordering) operations, the session uses the runtime area to perform all those memory-intensive operations.

■ **Note** A cursor is a handle to a private SQL area in memory, and the `OPEN_CURSORS` initialization parameter determines the number of cursors in your instance.

To reduce response time, all the sorts that are performed in the PGA should be performed completely in the cache of the work area—this is known as an *optimal mode operation*, since all work is done in memory, with no disk I/O whatsoever. If the sort operation spills onto the disk because the memory areas aren't adequate, that will slow down the sort operation. A SQL operation that is forced to use the disk area in a limited fashion is a *single-pass operation*, and it leads to slower performance than when the operation executes entirely in the memory cache. However, if your runtime memory area is too small relative to the sorting operation, Oracle will have to conduct multiple passes over the data being sorted, which is very disk intensive, and will result in extremely slow response times for the user. Thus, there is a direct correlation between the PGA size and query performance.

■ **Caution** Many Oracle manuals suggest that you can allocate up to half of the total system memory for the Oracle SGA. This guideline assumes that the PGA memory will be fairly small. However, if the number of users is very large and the queries are complex, your PGA component may end up being even larger than the SGA. You should estimate the total memory requirements by projecting both SGA and PGA needs.

You can tune the size of these private work areas, but this is a hit-or-miss approach that involves weighing a number of complex Oracle configuration parameters related to the work areas. The parameters that you need to manually configure include the `SORT_AREA_SIZE`, `HASH_AREA_SIZE`, and `BITMAP_AREA_SIZE` parameters.

The sum of all the PGA memory used by all sessions makes up the PGA used by the instance. Oracle recommends that you use *automatic PGA management*, which automates the allocation of PGA memory. This helps you use the memory allocated to your database more efficiently. The feature performs especially well when you have varying workloads, because it dynamically adjusts its available memory bounds and the work profiles on a continuous basis. Manual management of PGA could easily lead either to too little or too much memory being allocated, which causes severe performance problems.

You automate PGA memory allocation by ensuring that the `WORKAREA_SIZE_POLICY` initialization parameter is set to its default value of `auto`. If you set the parameter value to `manual`, you'll have to specify all the PGA work area–related parameters mentioned previously. The `WORKAREA_SIZE_POLICY` parameter ensures the automation of PGA memory. However, you must also set the size of the total PGA memory allocation by specifying a value for the `PGA_AGGREGATE_TARGET` initialization parameter. For example, if you set `PGA_AGGREGATE_TARGET=5000000000` in your initialization parameter file, Oracle uses the 5GB PGA allocation as a global target for the instance. Oracle will try to keep the total PGA memory used by all server processes attached to the instance under this target value.

If you don't set a value for the `PGA_AGGREGATE_TARGET` parameter, you'll be using the manual mode to manage the work areas. Alternatively, you can activate the manual mode by setting the `WORKAREA_SIZE_POLICY` parameter to `manual`. Oracle strongly recommends using automatic PGA management because it enables much more efficient use of memory. For users, this means better throughput and faster response time for queries in general.

■ **Note** In a manual management mode, any PGA memory that isn't being used isn't automatically returned to the system. Every session that logs into the database is allocated a specific amount of PGA memory, which it holds until it logs off, no matter whether it's performing SQL operations or not. Under automatic PGA management, the Oracle server returns all unused PGA memory to the operating system. On a busy system, this makes a huge difference in database and system performance. Suppose you set the `PGA_AGGREGATE_TARGET` parameter to 5GB. Oracle will not immediately grab all of the 5GB when you start the instance, as it does in the case of the `SGA_TARGET` parameter. It will only take the memory as necessary from the system, subject to the limit of 5GB. As soon as a session releases the run-area memory, the memory is automatically released to the operating system.

When you use automatic PGA memory management by setting the `PGA_AGGREGATE_TARGET` parameter, Oracle will do its best to assign enough memory to all work areas so they work in an optimal manner, executing all memory-intensive SQL operations in the cache memory. At worst, some work areas will use the disk areas in a single-pass mode. However, if you set the `PGA_AGGREGATE_TARGET` parameter too low relative to the work area needs of your instance, Oracle will be forced to conduct multi-pass executions of the sort- or hash-intensive SQL operations, with disastrous results for your instance performance.

I discuss PGA management in more detail in Chapter 22, which deals with tuning instance performance, and I show how to determine the optimal size for your `PGA_AGGREGATE_TARGET` initialization parameter.

A Simple Oracle Database Transaction

So far in this chapter, you've seen the components of the Oracle database system: the necessary files and memory allocations and how you can adjust them. It's time now to look into how Oracle processes users' queries and how it makes changes to data. It's important to understand the mechanics of SQL transaction processing because all interaction with an Oracle database occurs either in the form of SQL queries that read data or SQL (or PL/SQL) operations that modify, insert, or delete data.

A transaction is a logical unit of work in an Oracle database, and consists of one or more SQL statements. A transaction begins with the first executable SQL statement and terminates when you commit or roll back the transaction. Committing a transaction will make your changes permanent, and rolling back the changes will, of course, undo them. Once you commit the transaction, all other users' transactions that start subsequently will be able to see the changes made by your transactions.

When a transaction fails to execute completely (say, due to a power failure), the entire transaction must be undone. Oracle will roll back any changes made by earlier SQL statements in the transaction, leaving the data in its original (pre-transaction) state. The whole process is designed to maintain *data consistency*—a transaction is an all or nothing concept.

The following simple example of a row being inserted outlines how Oracle processes transactions:

1. A user requests a connection to the Oracle server through a 3-tier or an *n*-tier web-based client using Oracle Net Services.
2. Upon validating the request, the server starts a new dedicated server process for that user.
3. The user executes a statement to insert a new row into a table.
4. Oracle checks the user's privileges to make sure the user has the necessary rights to perform the insertion. If the user's privilege information isn't already in the library cache, it will have to be read from disk into that cache.
5. If the user has the requisite privileges, Oracle checks whether a previously executed SQL statement that's similar to the one the user just issued is already in the shared pool. If there is, Oracle executes this version of the SQL; otherwise Oracle parses and executes the user's SQL statement. Oracle then creates a private SQL area in the user session's PGA.
6. Oracle first checks whether the necessary data is already in the data buffer cache. If not, the server process reads the necessary table data from the data files on disk.
7. Oracle immediately applies row-level locks, where needed, to prevent other processes from trying to change the same data simultaneously.
8. The server writes the change vectors to the redo log buffer.
9. The server modifies the table data (inserts the new row) in the data buffer cache.
10. The user commits the transaction, making the insertion permanent. Oracle releases the row locks after the commit is issued.
11. The log writer process immediately writes out the changed data in the redo log buffers to the online redo log file.

12. The server process sends a message to the client process to indicate the successful completion of the INSERT operation. (If it couldn't complete the request successfully, it sends a message indicating the failure of the operation.)
13. Changes made to the table by the insertion may not be written to disk right away. The database writer process writes the changes in batches, so it may be some time before the inserted information is actually written permanently to the database files on disk.

■ **Note** In the previous example, since a new row is being inserted, there is no undo information to record in the undo tablespace. If the user had updated a row instead, Oracle would have had to record the before-update row in the undo records. Until the original transaction commits the update, all other users will see the original data values of the row.

Data Consistency and Data Concurrency

Databases aren't very useful if a large number of users can't access and modify data simultaneously. *Data concurrency* refers to the capability of the database to handle this concurrent use by many users. To provide consistent results, the database also needs a mechanism within it that ensures users don't step on each other's changes. *Data consistency* refers to the ability of a user to get a meaningful and consistent view of the data, including all the changes made to it by other users.

Oracle uses special structures called *undo segments* to ensure data consistency. For example, when you're reading a set of data for a transaction, Oracle ensures that the data you read is *transaction-set consistent*; that is, it guarantees that the data you see reflects a single set of committed transactions. Oracle also provides *read consistency* of data, meaning that all the data selected by your queries comes from a single point in time. Oracle's undo segments are part of the undo tablespace mentioned earlier in this chapter.

Oracle uses locking mechanisms to ensure data concurrency. By allowing one user to lock individual rows or entire tables, that user is guaranteed exclusive use of the table for updating purposes. An important feature of the Oracle locking mechanisms is that they are, for the most part, automatic. You don't need to concern yourself with the details of how to lock the objects you want to modify—Oracle will take care of it for you behind the scenes.

Oracle uses two basic modes of locking. The *exclusive lock mode* is used for updates, and the *share lock mode* is used for SELECT operations on tables. The share lock mode enables several users to simultaneously read the same rows in a table. The exclusive lock mode, because it involves updates to the table, can only be used by one user at any given time. Exclusive locks are almost always applied to the specific rows being updated, permitting simultaneous use of the database by several users. Oracle releases the locks it holds on the tables and other internal resources automatically after the issue of a COMMIT, SAVEPOINT, or ROLLBACK command.

Oracle locking is complex, and you'll learn about it in detail in Chapter 6, along with how Oracle ensures data consistency and concurrency.

The Database Writer and the Write Ahead Protocol

The database writer, as you saw earlier, is responsible for writing all modified buffers in the database buffer cache to the data files. Further, it has the responsibility of ensuring there is free space in the buffer cache so the server process can read in new data from the data files when necessary. The

(log) *write ahead protocol* also requires that the redo records in the redo log buffer associated with the changed data in the data buffer cache are written to the redo log data files before the changes are recorded in the data files themselves. The importance of the redo log contents makes it imperative that Oracle write the contents of the redo log file to permanent storage before it writes the changes to the data files on disk.

When users commit their transactions, the log writer process immediately writes only a single commit record to the redo log files. The entire set of records affected by the committed transaction may not be written simultaneously to the data files. This *fast commit mechanism*, along with the write ahead protocol, ensures that the database is not kept waiting for all the physical writes to be completed after each transaction. As you can well imagine, a huge OLTP database with numerous changes throughout the day cannot function optimally if it has to write to disk after every committed data change.

■ **Note** If there are a large number of transactions and, therefore, a large number of commit requests, the log writer process may not write each committed transaction's redo entries to the redo log immediately. It may batch multiple commit requests if it is busy writing previously issued commit records. This batched writing of redo entries from multiple committed transactions is known as *group commits*.

The System Change Number

The *system change number* (SCN) is an important quantifier that the Oracle database uses to keep track of its state at any given point in time. When you read (`SELECT`) the data in the tables, you don't affect the state of the database, but when you modify, insert, or delete a row, the state of the database is different from what it was before. Oracle uses the SCN to keep track of all the changes made to the database over time. The SCN is a *logical time stamp* that is used by Oracle to order events that occur within the database. The SCN is very important for several reasons, not the least of which is the recovery of the database after a crash.

SCNs are like increasing sequence numbers, and Oracle increments them in the SGA. When a transaction modifies or inserts data, Oracle first writes a new SCN to the rollback segment. The log writer process then writes the commit record of the transaction immediately to the redo log, and this commit record will have the unique SCN of the new transaction. In fact, the writing of this SCN to the redo log file denotes a committed transaction in an Oracle database.

The SCN helps Oracle determine whether crash recovery is needed after a sudden termination of the database instance or after a `SHUTDOWN ABORT` command is issued. Every time the database checkpoints, Oracle writes a `START SCN` command to the data file headers. The control file maintains an SCN value for each data file, called the `STOP SCN`, which is usually set to infinity, and every time the instance is stopped normally (with the `SHUTDOWN NORMAL` or `SHUTDOWN IMMEDIATE` command), Oracle copies the `START SCN` number in the data file headers to the `STOP SCN` numbers for the data files in the control file. When you restart the database after a graceful shutdown, there is no need for any kind of recovery because the SCNs in the data files and the control files match. On the other hand, abrupt instance termination does not leave time for this matching of SCNs, and Oracle recognizes that instance recovery is required because of the varying SCN numbers in the data files on the one hand and the control file on the other. As you'll learn in Chapter 16, they play a critical role during database recovery. Oracle determines how far back you should apply the archived redo logs during a recovery based on the SCN.

Undo Management

When you make a change to a table, you should be able to undo or roll back the change if necessary. The information needed to undo or roll back changes in transactions, which mostly consists of the pre-change table row information, is called *undo data* (the change vectors), and it is stored in *undo records*. When you issue a `ROLLBACK` command, Oracle uses these undo records to replace the changed data with the original versions. As Chapter 6 explains in detail, the undo records are vital during database recovery when all unfinished or uncommitted transactions must be discarded to make the database consistent.

In earlier versions of the Oracle database (up to Oracle8*i*), it was the DBA's job to manage what are known as *rollback segments* by explicitly allocating a regular, permanent tablespace for them. In fact, the management of the rollback segments used to be a vexing and time-consuming part of the job for many DBAs who managed large databases, especially if they had frequent, long-running transactions. Oracle wrote (and still writes) to the rollback segments in a circular fashion, and it wasn't uncommon to find that information needed by a transaction for read consistency had been overwritten by a newer transaction. Many DBAs used to find the sizing of the rollback segments a tricky issue: if you had several small rollback segments, a large transaction might fail, and if you had a small number of very large rollback segments, your transactions might encounter contention for the rollback segment transaction tables.

The manual mode of undo management is deprecated in Oracle Database 10*g*. Oracle strongly recommends the use of the *Automatic Undo Management* (AUM) feature, where the Oracle server itself will maintain and manage the undo (rollback) segments. All you need to do is provide a dedicated undo tablespace and set the initialization parameter `UNDO_MANAGEMENT` to auto. The default setting of the `UNDO_MANAGEMENT` parameter is still `manual` in Oracle Database 10*g*—Oracle will create the necessary number of undo segments, which are structurally similar to the traditional rollback segments, and it'll size and extend them as necessary. It's not uncommon for new undo segments to be created and old ones to be deactivated based on the amount of transactions going on in the database. Chapter 6 provides further information about the AUM feature.

Because Oracle will do the sizing of the individual undo segments for you, the two decisions you have to make are the size of the undo tablespace and the setting for the `UNDO_RETENTION` initialization parameter (which determines how long Oracle will try to retain undo records in the undo tablespace). Remember that your undo tablespace should not only be able to accommodate all the long-running transactions, but it also has to be big enough to accommodate any *flashback* features you may implement in your database—Oracle's flashback features let you undo changes to data at various levels. Several flashback features, such as Flashback Query, Flashback Versions Query, and Flashback Table, utilize undo data. I discuss the undo-related Flashback features in Chapter 6.

You can use Oracle's Undo Advisor through the OEM to figure out the ideal size for your undo tablespaces and the ideal duration to specify for the `UNDO_RETENTION` parameter. Using the current undo space consumption statistics, you can estimate future undo generation rates for the instance.

Backup and Recovery Architecture

You must perform regular backups of any database that contains useful information. All databases depend on mechanical components like disk drives, and they are also subject to unexpected events like power failures and natural catastrophes. Programmatic and user errors also necessitate protecting data through a strong backup system. Recovery involves two main objectives: first, you must return the database to a normal operating state with as little downtime as possible. Second, you mustn't lose any useful data.

It's important to understand the basics of how Oracle manages its backup and recovery operations. You've seen some of the components previously, but I put it all together here. The following Oracle structures ensure that you can recover your databases after a problem:

- *The control file*: The control file contains data file and redo log information, as well as the latest system change number, which is key to the recovery process.
- *Database backups*: These are file or tape backups of the database data files. Since these backups are made periodically, they most likely won't contain all the data needed to bring the database up to date.
- *The redo logs*: The redo logs, as you've seen earlier in this chapter, contain all changes made to the database, including uncommitted and committed changes.
- *The undo records*: These records contain the before-images of blocks changed in uncommitted transactions.

Recovery involves restoring all backups first. Since the backups can't bring you up to date, you apply the redo logs next, to bring the database up to date. Since the redo logs may contain some uncommitted data that shouldn't really be in the database, however, Oracle uses the undo records to roll back all the uncommitted changes. When the recovery process is complete, your database will not have lost any committed or permanent data.

User-Managed Backup and Recovery

You can perform all backup and recovery procedures by issuing direct commands through SQL*Plus and operating system commands. However, Oracle strongly recommends that you use the Oracle-provided Recovery Manager (RMAN) to perform your backup and restore work.

RMAN

RMAN is Oracle's main backup and recovery tool. You can use RMAN from the command line as well as through a GUI interface. RMAN enables various types of backup and recovery techniques, and several of these techniques are unique to the tool. For example, a big benefit of using RMAN is that it automatically maintains all records of existing database backups, without you having to maintain that information somewhere.

The Automatic Disk-Based Backup and Recovery feature uses a flash recovery area to help you automate the management of backup-related files. Oracle recommends that you use such a flash recovery area, which is a location on disk where the database stores and manages all recovery-related files, like archived redo logs and other files for your database. Files no longer needed in the flash recovery area are deleted automatically when RMAN needs to reclaim space for new files. If you don't use a flash recovery area, you must manually manage disk space for your backup-related files.

Oracle Backup

RMAN can't back up files directly to tape devices, and in previous versions of Oracle you had to use a third-party tool (for example, NetWorker from Legato) to manage tape backups. In Oracle Database 10*g* Release 2, you have access to the exciting Oracle Backup feature, which is an out-of-the-box backup and recovery solution for Oracle customers. Oracle Backup copies data files directly to tape and manages the archiving of those tapes as well. Chapter 15 provides an introduction to Oracle Backup.

You can easily configure the Backup Manager through OEM. By using OEM and Oracle Backup together, you can easily back up and recover databases enterprise-wide.

Flashback Recovery Techniques

Quite often, you may be called upon to help recover from a logical corruption of the database, rather than from a hardware failure. You can use the following flashback techniques in Oracle Database 10*g* to recover from logical errors:

- *Flashback Database*: Takes the entire database back to a specific point in time
- *Flashback Table*: Returns individual tables to a past state
- *Flashback Drop*: Undoes a `DROP TABLE` command and recovers the dropped table
- *Flashback Query, Flashback Version Query, and Flashback Transaction Query:* Retrieve data from a time (or interval) in the past

The Flashback Database and Flashback Drop features are discussed in Chapter 16, which deals with recovery techniques. The Flashback Table, Flashback Query, Flashback Version Query, and Flashback Transaction Query features rely on undo data, and are covered in Chapter 6.

The Oracle Data Dictionary and the Dynamic Performance Views

Oracle provides a huge number of internal tables to aid you in tracking changes to database objects and to fix problems that will occur from time to time. Mastering these key internal tables is vital if you want to become a savvy Oracle DBA. All the GUI tools, such as OEM, depend on these key internal tables (and views) to gather information for monitoring Oracle databases. Although you may want to rely on GUI tools to perform your database administration tasks, it is important to learn as much as you can about these internal tables. Knowledge of these tables helps you understand what is actually happening within the database.

You can divide the internal tables into two broad types: the static data dictionary tables and the dynamic performance tables. You won't access these tables directly; rather, you'll access the information through views based on these tables. Chapter 23 is dedicated to a discussion of these views, and you can get a complete list of all the data dictionary views by issuing the following simple query:

```
SQL> SELECT * FROM dict;
```

The following sections examine the role of these two important types of tables (and views).

The Oracle Data Dictionary

Oracle maintains a set of tables within the database called the *data dictionary.* You access these read-only data dictionary tables through views built on them. Views are like logical tables built on an underlying Oracle table, and I discuss them in detail in Chapter 5. The data that the data dictionary maintains is also known as *metadata.* DBAs and developers depend heavily on the data dictionary for information about the various components of the database—these tables contain information such as the list of tables, table columns, users, user privileges, file and tablespace names, and so on. A simple query, such as the following, necessitates several calls to the data dictionary before Oracle can execute it:

```
SQL> SELECT employee_name
     FROM emp
     WHERE city = 'NEW YORK';
```

It's important to note that the data dictionary tables don't report on aspects of the running instance. The data dictionary holds only information about the database, such as the database files,

tables, functions, and procedures, as well as user-related information. Another set of views, called the *dynamic performance views*, records information about the currently running instance.

■ **Tip** The data dictionary tables describe the entire database: its logical and physical structure, its space usage, its objects and their constraints, and user information. You can't access the data dictionary tables directly; instead, you're given access to views built on them. You also can't change any of the information in the data dictionary tables yourself. Only Oracle has the capability to change data in the data dictionary tables. When you issue a query involving the `CITIES` column in a table named EMPLOYEES, for example, the database will consult various data dictionary tables to verify that the table and the column exist, and to confirm that the user has the rights to execute that statement. As you can imagine, a heavily used OLTP database will require numerous queries on its data dictionary tables during the course of a day.

The Oracle super user SYS owns most of the data dictionary tables (though some are created under the system username), and they are stored in the System tablespace.

■ **Tip** Oracle recommends that you analyze both the data dictionary and the dynamic performance tables (also referred to as fixed tables) on a regular basis to improve performance. Chapter 21 shows you how to analyze these tables.

The Dynamic Performance (V$) Views

In addition to the data dictionary, Oracle maintains an important set of *dynamic performance tables*. These tables maintain information about the current instance, and Oracle continuously updates these tables.

The set of virtual dynamic tables is referred to as the *X$ tables*. Oracle doesn't allow you to access the X$ tables directly; rather, Oracle creates views on all these tables and then creates synonyms for these views. You'll be accessing these views, called the *V$ views*, to get information about various aspects of a running instance. The V$ views are the foundation of all Oracle database performance tuning. If you wish to master the Oracle database, you must master the V$ dynamic views, because they are the wellspring of so much knowledge about the Oracle instance.

The dynamic performance views, like the data dictionary views, are based on read-only tables that only Oracle can update. Some of the tables capture session-wide information, and some of them capture system-wide information. You'll find the dynamic views extremely useful in session management, backup operations and, most important, in performance tuning. Remember, though, that the dynamic performance tables are only populated for the duration of the instance and are cleaned out when you shut down the instance.

The Oracle Optimizer

In most cases, when users issue a query against the database, there's more than one way to access the tables and retrieve the data. Because there are many ways to execute the same statement, Oracle uses a *cost-based optimizer* (CBO) to choose the best execution plan for queries, based on the cost of the query in terms of resource use.

Query optimizing is at the heart of modern relational databases and is an essential part of how Oracle conducts its operations. The query optimizer is transparent to users and Oracle will automatically apply the best access and join methods to your queries before it starts processing.

■ **Note** To choose the best execution plans, Oracle uses statistics on tables and indexes, which include counts of the number of rows and the data distribution of "data skew" in the tables within the database. (The physical storage statistics and the data distribution statistics for all database tables and indexes, columns, and partitions are stored in various data dictionary tables.) Armed with this information, the optimizer usually succeeds in finding the best path to access the necessary data for executing a SQL statement. Oracle also lets you use *hints* to override the optimizer's choice of an execution path. This is because in some instances the application developer's knowledge of the data enables the use of more efficient execution plans than the optimizer can come up.

I discuss the Oracle optimizer in detail in Chapter 21, in the context of performance tuning.

■ **Tip** In Oracle Database 10*g*, you can also use the Oracle optimizer in an enhanced tuning mode, as shown in Chapter 21. The Oracle optimizer in the tuning mode is the basis of the new SQL Tuning Advisor feature, also explained in Chapter 21.

Talking to the Database

In order for a user to communicate with the database, he or she must first connect to the database by creating a user session. The user communication with the database is done through one of several interfaces. This section will quickly review Oracle database connectivity aspects and the main communication interfaces, including SQL*Plus, iSQL*Plus, and the OEM Database Control and Grid Control interfaces, which serve as the main DBA management consoles.

Connecting to Oracle

You can connect to the Oracle database from the server on which the Oracle RDBMS is running. However, DBAs as well as application developers and users generally connect to the database through the network using Oracle Net, a component of Oracle Net Services. Oracle Net enables network sessions from a client application to an Oracle database server. It acts as the data courier for the clients and the database server, and it is in charge of establishing and maintaining the connection as well as transmitting messages between client and server. Oracle Net is installed on each computer in the network.

■ **Note** Oracle Net Services is Oracle's mechanism for interfacing with the communication protocols (TCP/IP, FTP, and so on) that define the way data is transmitted and received on a network.

For a connection to succeed, the client application must specify the location of the database. On the database side, the Oracle Net listener, known simply as the *listener*, is the process that listens for incoming client connection requests. You configure the listener in the listener.ora file, where you provide the database address. The listener.ora file also defines the protocol the listener is listening on, and related information. On the client side, you can either use the tnsnames.ora file to list the database server connection details, which include the database name, server name, and the connection protocol, or you can use the newer and much simpler *easy connect method* in Oracle Database 10*g*.

Oracle Enterprise Manager (OEM)

Oracle Enterprise Manager (OEM) is Oracle's GUI-based management tool that lets you manage one or more databases efficiently. OEM enables security management, backups, and routine user and object management. Because OEM is GUI-based, you don't have to know a lot of SQL to use the tool. However, understanding the V$ and dynamic performance views will enhance your knowledge of how the database works—OEM will be an even more powerful tool in your hands after you master the management of the database using the data dictionary–based and dynamic performance table–based SQL queries. Oracle has really improved OEM in its most recent versions, and all serious practitioners of the trade should master the use of the tool for both daily database management as well as scheduling routine database administration tasks and troubleshooting. Chapter 19 explains the configuration and use of the OEM tool set.

In Oracle Database 10*g*, you have the option of using either the Database Control or Grid Control version of Enterprise Manager. Enterprise Manager Database Control is automatically installed along with the Oracle software and is designed to run as a stand-alone application. In order to manage several databases, however, you need to separately install the Enterprise Manager Grid Control software on your server and the OEM Agent software on all the targets you wish to monitor.

The Oracle Enterprise Manager tool always looked promising in previous versions, but it delivered inconsistent performance. This hard reality, plus the fact that many DBAs are comfortable with manual commands and scripts based on the database dictionary and the dynamic (V$) views, led to a low acceptance rate of the tool. In Oracle Database 10*g*, the OEM tool has gone through a sea change and delivers high-level performance. I strongly recommend using the Database Control or the Grid Control tool to monitor and manage your databases. You can invoke all the new management advisors and tools, like the ADDM from the OEM toolset, without having to use complex Oracle PL/SQL packages. I show OEM examples throughout this book.

■ **Note** Traditionally, all GUI tools relied on the same V$ performance views that are used in database queries. In Oracle Database 10*g* Release 2, however, OEM can access key performance data directly from the SGA, without making any SQL queries. This is done by attaching directly to the SGA and reading the statistics from the shared memory. When your database is performing extremely slowly or hangs, you can't rely on the dynamic V$ views to troubleshoot the problem—doing so may actually end up making matters worse! This is one more reason why you should make the OEM your main means of monitoring and managing the Oracle instance.

SQL*Plus

SQL*Plus is an Oracle tool that lets you enter and run SQL statements and PL/SQL (a procedural extension to the Oracle SQL language) blocks. As a DBA, you can perform all your tasks right from the SQL*Plus interface itself. However, as I explain in the previous "Oracle Enterprise Manager" section, you may want to make the SQL*Plus interface your secondary, rather than primary, tool for accessing the Oracle RDBMS. SQL*Plus is discussed more in Chapter 12.

*i*SQL*Plus

*i*SQL*Plus is a browser-based interface to the Oracle database, and it is very similar to SQL*Plus. It generates its output in the form of HTML tables, and you don't need to install or configure anything for the *i*SQL*Plus user interface other than a web browser. On the server side, only an Oracle HTTP server with the *i*SQL*Plus Server is needed. Chapter 12 shows how to use the iSQL*Plus interface.

Oracle Utilities

Oracle provides several powerful tools to help with loading and unloading of data and similar activities. The following sections describe the main ones.

Data Pump Export and Import

The Data Pump Export and Import utilities are the successors to the traditional export and import utilities; they help with fast data loading and unloading operations. The original export and import utilities are still available, but Oracle recommends the use of the newer and more sophisticated tools. Chapter 14 discusses the Data Pump utility in detail.

SQL*Loader

The SQL*Loader is a powerful and fast utility that loads data from external files into tables of an Oracle database. Chapter 13 discusses SQL*Loader in detail.

External Tables

You use the SQL*Loader to load external data into an Oracle table. Sometimes, though, you need to use some external data but don't want to go to the trouble of loading the data into a table. The *external tables* feature offers some of the SQL*Loader utility's functionality.

External tables let you use data that resides in external text files as if it were in a table in an Oracle database. In Oracle Database 10*g*, you can *write* to external tables as well as read from them. External tables are dealt with in detail in Chapter 13.

LogMiner

The LogMiner utility lets you query online and archived redo log files through a SQL interface. As you know, redo log files hold the history of all changes made to the database. Thus, you can use the LogMiner to see exactly which transaction and what SQL statement caused a change, and if necessary, undo it. Chapter 16 shows you how to use the LogMiner tool for precision recovery.

Scheduling and Resource-Management Tools

Oracle Database 10*g* provides several utilitarian tools for scheduling jobs and managing database and server resource usage, and they're outlined in the following sections.

The Scheduler

The new Scheduler facility lets DBAs schedule tasks from within the Oracle database, without having to write shell scripts and scheduling them through the operating system. The Oracle Scheduler feature uses functions and procedures of the new DBMS_SCHEDULER package.

The basic components of the Oracle Scheduler are jobs, programs, and schedules. The Oracle Scheduler offers much more functionality than using the old DBMS_JOBS package. You can now create common jobs and schedules that you can share across users. You can also group similar jobs into job classes and use resource plans to prioritize resources among resource consumer groups. You can schedule PL/SQL and Java programs as well as operating system shell scripts through the Scheduler.

You'll find a complete treatment of the Oracle Scheduler in Chapter 18.

Database Resource Manager

The Database Resource Manager lets you exercise control over how the server resources, especially CPU resources, are allocated among your users. You first group the users according to common resource requirements, and you then create directives that dictate how resources are to be allocated to these groups. The Database Resource Manager controls how long the sessions run, thus ensuring that resource usage matches the stated objectives. I discuss the Database Resource Manager in detail in Chapter 11.

Automatic Database Management

Traditionally, Oracle DBAs had to exercise great care in setting numerous initialization parameters, and they would spend quite a bit of their time tweaking those parameters, trying to achieve an ideal database configuration. Oracle started a major push toward a self-managing database with the 9*i* version, and Oracle Database 10*g* takes that effort further, offering a more complete set of self-managing features, especially in the performance-tuning area. In the long run, the goal is to automate all routine tasks and free up the DBAs and other professionals to use their time to further the strategic interests of their businesses.

The following sections summarize the main automatic management features in Oracle Database 10*g*.

Automatic Database Diagnostic Monitor (ADDM)

The Automatic Database Diagnostic Monitor (ADDM) is probably the most revolutionary aspect of the new self-managing Oracle database. The ADDM is a diagnostic engine built right into the database kernel—it is a rule-based expert system that encapsulates decades of Oracle's performance-tuning expertise. It analyzes performance data frequently and either makes a recommendation by itself, or suggests that you invoke one of the other Oracle advisory components, such as the SQL Tuning Advisor.

The ADDM proactively performs automatic monitoring of the database at regular intervals throughout the day, performs a top-down analysis of performance data and bottlenecks, and presents a set of findings that include the root causes of problems and the recommendations to fix them. In addition, it provides the rationale behind its recommendations. Because the ADDM quantifies the identified problems in terms of their impact on overall performance, you can focus on fixing problems that will give you the biggest performance gains.

You can also run the ADDM manually through the Enterprise Manager or the command line. The ADDM's diagnostic abilities can be used during the development phase of applications, reducing potential problems in production. The ADDM will enable developers to perform "what-if" tests very easily.

Chapter 17 explains the ADDM in detail.

Automatic Memory Management

You can turn on automatic shared memory management and have the database manage the SGA for you; simply set the `SGA_TARGET` initialization parameter to a nonzero value. Oracle will manage the shared pool, buffer cache, large pool, Java pool, and streams pool memory components automatically. This will eliminate the trial and error method of determining optimal SGA allocation.

Similarly, you can enable automatic PGA management and have Oracle determine the optimal PGA memory allocation; set the `PGA_AGGREGATE_TARGET` initialization parameter. When your database performs a lot of sorting and hashing operations, automatic PGA management is critical in achieving peak performance. I discuss automatic PGA management in more detail in Chapter 22.

Automatic Undo Retention Tuning

Setting the `UNDO_RETENTION` parameter to zero or just leaving it out of your SPFILE will instruct Oracle to perform proactive automatic undo retention tuning, thus reducing the occurrence of the well-known "snapshot too old" errors that lead to the failure of many an overnight production batch job. Under automatic undo retention tuning, Oracle will figure out the ideal retention period for undo data, based on the length of the transactions and other related factors. I discuss automatic undo retention tuning in Chapter 6.

Automatic Checkpoint Tuning

By setting the `FAST_START_MTTR_TARGET` initialization parameter to a nonzero value, or by not setting it at all, you can automate checkpoint tuning. This means that you won't have to set any checkpoint initialization parameters, telling Oracle how frequently it should perform database checkpointing. This will help your database recover in a reasonable length of time following a crash. Chapter 17 reviews automatic checkpoint tuning.

Automatic Optimizer Statistics Collection

Oracle Database 10*g* automatically gathers statistics for the cost-based optimizer through a regularly scheduled job. The job gathers statistics on all objects in the database that have missing or stale statistics. Oracle creates this job automatically at database creation time, and the Scheduler automatically manages it. Chapter 17 discusses the automatic collection of optimizer statistics.

Automatic Storage Management (ASM)

Automatic Storage Management (ASM) is the new Oracle Database 10*g* feature that integrates your file system with a volume manager that's designed for Oracle files. ASM divides Oracle data files into extents, which it distributes evenly across the disk system. ASM automatically redistributes I/O load across all available disks whenever storage configuration changes, avoiding manual disk tuning. ASM also provides mirroring and striping, thus enhancing protection and performance, as in RAID systems. ASM is dealt with in detail in Chapter 17.

Automatic Segment Advisor Operation

In Oracle Database 10*g* Release 2, the Segment Advisor runs automatically during the nightly maintenance job of the Oracle Scheduler, and checks whether your database has any wasted space that can be reclaimed by shrinking segments such as tables and indexes. The advisor may recommend either a segment shrink or a reorganization operation, depending on whether the tablespace is locally managed or not (it will recommend shrinking the object if it is locally managed and reorganizing it otherwise). DBAs spend a lot of time reorganizing their database objects, and this is a wonderful way to cut back on all that effort and time. Chapter 17 shows you how to use the Segment Advisor.

Common Manageability Infrastructure

In order to be self-tuning and self-managing, a database must have the ability to automatically "learn" how it is being used. To this end, Oracle provides a common manageability infrastructure, which captures workload information and uses it to make sophisticated self-management decisions. The heart of the manageability infrastructure is the new *Automatic Workload Repository*

(AWR), which serves as a repository for all the other server components that aid automatic management of the database.

Oracle has built instrumentation into the various layers of its technology stack to capture the metadata that helps in diagnosing performance. It stores this data in the AWR and utilizes a comprehensive suite of management advisors to provide guidance on optimizing database operations. In the following sections, I briefly explain the various components of the common manageability infrastructure of Oracle Database 10*g*. You'll fully explore all of these in later chapters.

Automatic Workload Repository (AWR)

The AWR plays the role of the "data warehouse of the database," and it is the basis for most of Oracle's self-management functionality. The AWR collects and maintains performance statistics for problem-detection and self-tuning purposes. By default, every 60 minutes the database collects statistical information from the SGA and stores it in the AWR, in the form of snapshots.

Several database components, such as the ADDM and other management advisors, use the AWR data to detect problems and for tuning the database. Like the ADDM, the AWR is automatically active upon starting the instance. You'll learn more about the AWR in Chapter 18.

Active Session History (ASH)

In Oracle Database 10*g*, active sessions are sampled every second, and the session information is stored in a circular buffer in SGA. A session that's either waiting for a non-idle event or that was on the CPU is considered an active session.

Even though the ADDM provides you with detailed instance information by periodically analyzing the AWR data, you are at a loss if you want to know what's happened in the database in a recent time period (such as in the past five minutes). Active Session History (ASH) and its related historical views provide you with insight into current activity in the database. Chapter 18 discusses ASH in detail.

Server-Generated Alerts

Oracle now sends out proactive server-generated alerts to warn you about problems like a tablespace running out of space. You can configure server-generated alerts by setting warning and critical thresholds on database metrics. The Oracle server automatically alerts you, for example, when the physical database reads per second cross a preset threshold value, or when a tablespace is low on free space. Server-generated alerts are discussed in Chapter 18.

Automated Tasks Feature

Oracle automatically performs certain maintenance tasks, such as collecting optimizer statistics, by using the new Scheduler feature. Oracle keeps track of which database objects don't have statistics or have stale statistics, and automatically refreshes statistics for these objects. In previous versions of Oracle, the DBA was responsible for collecting up-to-date statistics on all objects in the database. Now the database itself manages the collection of these statistics. Automated tasks are discussed in detail in Chapter 18.

■ **Note** The manageability infrastructure, as well as all the automatic management features, are installed when you install the Oracle Database 10*g* software.

Advisory Framework

Oracle Database 10*g* comes with several management advisors, which help tune your SQL queries, size your memory and undo configuration parameters, and figure out the right indexes and materialized views for your database. The advisors use a uniform interface—the Advisor Central in the OEM, or the DBMS_ADVISOR package, when you invoke them manually. All the advisors use the Automatic Workload Repository as the source of their data and as a repository for their reports. Chapter 18 introduces the advisory framework in detail. Here's a brief description of the main management advisors, which you'll see in detail in later chapters.

SQL Tuning Advisor

The SQL Tuning Advisor provides recommendations for running SQL statements faster, by replacing manual tuning with tuning suggested by the Automatic Tuning Optimizer, which is the Cost Optimizer in a tuning mode. The advisor calls the Automatic Tuning Optimizer (ATO) to perform optimizer statistics analysis, SQL profiling, access-path analysis, and SQL structure analysis. The SQL Tuning Advisor is discussed in detail in Chapter 21.

SQL Access Advisor

The SQL Access Advisor provides advice on materialized views, indexes, and materialized view logs, in order to design the most appropriate access structures to optimize SQL queries. Chapter 5 shows you how to use the SQL Access Advisor.

Segment Advisor

Often, table segments become fragmented over time. The Segment Advisor checks database object space usage and helps you regain excess space in segments by performing segment-shrinking operations. The advisor also helps in predicting the size of new tables and indexes and analyzing database-object growth trends. Chapter 17 shows you how to use the Segment Advisor.

Undo Advisor

The Undo Advisor can recommend the correct size for your undo tablespaces and undo retention parameter, both of which are based on transaction volume and length. The advisor also can take into account undo requirements for supporting flashback features for a given length of time. Chapter 6 shows you how to use the Undo Advisor to get recommendations about the undo tablespace and the undo retention period.

SGA and PGA Memory Advisors

The SGA Memory Advisor recommends the ideal SGA size. The PGA Memory Advisor provides recommendations for the PGA parameter, based on the workload characteristics of the instance. Chapter 17 provides examples of the use of the SGA and PGA Memory Advisors.

Efficient Managing and Monitoring

You've seen a bewildering number of tools and components of management infrastructure for monitoring and managing your Oracle databases. Traditionally, DBAs used a variety of methods to manage and monitor their databases, and complaints about frequent midnight pages and weekend work were common. You can avoid all that by taking a proactive approach and by automating management as much as you can—and with Oracle Database 10*g*, you can automate quite a bit!

My advice is not to reinvent the wheel by using outmoded monitoring scripts and management techniques. Here's a suggested way to use Oracle's variety of tools to maximum benefit:

- Make the OEM Database Control or Grid Control your main DBA tool. You can access all the monitoring and performance tools through the OEM. Configure the OEM to send you event-based pages or e-mails.
- Use RMAN and Oracle Backup as your main database backup solutions.
- Configure the flash recovery area so you can automate backup and recovery.
- Use the Scheduler to automate your job system.
- Use locally managed tablespaces (automatic segment space management is the default, starting with Oracle Database 10*g* Release 2).
- Change your export and import scripts to the new Data Pump technology, both to save time and to take advantage of the new features.
- Wherever possible, use the Database Configuration Assistant (DBCA) to create new databases and the Database Upgrade Assistant (DBUA) to upgrade to Oracle Database 10*g* from earlier versions.
- Let Oracle automatically collect statistics—don't bother using the DBMS_STATS tool to manually collect optimizer statistics.
- Make sure you collect system statistics in addition to the automatic optimizer statistics collected by Oracle.
- Let Oracle manage the SGA and the PGA automatically.
- Automate both the undo management as well as checkpointing.
- Use Oracle's alert system to prevent space-related problems.
- Make use of the SQL Access Advisor to recommend new indexes and materialized views.
- Let the Segment Advisor, which runs automatically in Oracle Database 10*g* Release 2, recommend objects to shrink. Shrinking objects will reclaim unused and fragmented space, as well as decrease query response time.
- Use the SQL Tuning Advisor to tune problem SQL code.

Each of these topics is explained in detail in the rest of this book.