

# PART I

## Oracle Database 11g Administration

- **Chapter 1** Architectural Overview of Oracle Database 11g
- **Chapter 2** Installing and Creating a Database
- **Chapter 3** Instance Management
- **Chapter 4** Oracle Networking
- **Chapter 5** Oracle Storage
- **Chapter 6** Oracle Security

# CHAPTER 1

## Architectural Overview of Oracle Database 11g

### **Exam Objectives**

In this chapter you will learn to

- 052.1.1 Explain the Memory Structures
- 052.1.2 Describe the Process Structures
- 052.1.3 Identify the Storage Structures

This guide is logically structured to enable a thorough understanding of the Oracle server product and the fundamentals of SQL (Structure Query Language, pronounced *sequel*). The authors seek to relate your learning as much to the real world as possible to concretize some of the abstract concepts to follow, by introducing a hypothetical scenario that will be systematically expanded as you progress through the book. This approach involves nominating you as the DBA in charge of setting up an online store. You will appreciate the various roles a DBA is expected to fulfill as well as some of the technology areas with which a DBA is expected to be familiar.

The nonexaminable discussion of the Oracle product stack is followed by considering several prerequisites for fully understanding the tasks involved in setting up an Oracle 11g database system. This discussion leads into the examinable objectives in this chapter, which are the Single-Instance Architecture and the Memory, Process, and Storage Structures.

## Oracle Product Stack

No Oracle guide is complete without contextualizing the product under study. This section discusses the three core product families currently available from Oracle Corporation. End users of Oracle technology typically use a subset of the available products that have been clustered into either the server, development tools, or applications product families.

### Oracle Server Family

The three primary groupings of products within the server technology family consist of the database, application server, and enterprise manager suites. These form the basic components for Oracle's vision of grid computing. The concept underlying the Grid is *virtualization*. End users request a service (typically from a web-based application), but they neither know nor need to know the source of that service. Simplistically, the database server is accessible to store data, the application server hosts the infrastructure for the service being requested by the end user, and the enterprise manager product provides administrators with the management interface. The platforms or physical servers involved in supplying the service are transparent to the end user. Virtualization allows resources to be optimally used, by provisioning servers to the areas of greatest requirement in a manner transparent to the end user.

### Database Server

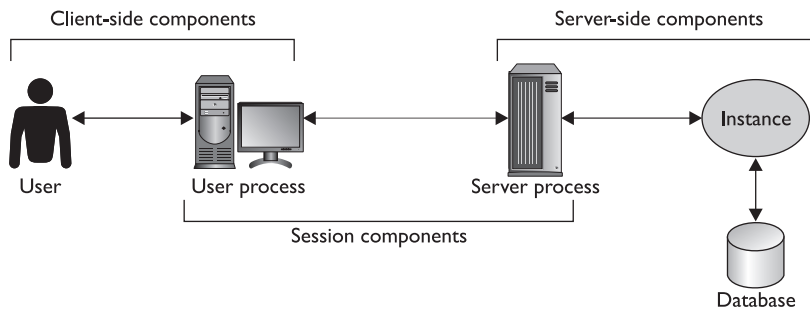
The database server comprises Oracle instances and databases with many features like Streams, Partitioning, Warehousing, Replication, and Real Application Clusters (RAC), but ultimately it provides a reliable, mature, robust, high-performance enterprise-quality data store, built on an object-relational database system. Historically, one of the projects undertaken in the late 1970s to animate the relational theory proposed by Dr. E.F. Codd resulted in the creation of a relational database management system (RDBMS) that later became known as the Oracle Server. The Oracle Server product is well established in the worldwide database market, and the product is central to

Oracle Corporation's continued growth, providing the backbone for many of its other products and offerings. This book is dedicated to describing the essential features of the Oracle Server and the primary mechanisms used to interact with it. It covers the aspects that are measured in the certification exams, but by no means explores the plethora of features available in the product.

An Oracle database is a set of files on disk. It exists until these files are deleted. There are no practical limits to the size and number of these files, and therefore no practical limits to the size of a database. Access to the database is through the Oracle instance. The *instance* is a set of processes and memory structures: it exists on the CPU(s) and in the memory of the server node, and its existence is temporary. An instance can be started and stopped. Users of the database establish sessions against the instance, and the instance then manages all access to the database. It is absolutely impossible in the Oracle environment for any user to have direct contact with the database. An Oracle instance with an Oracle database makes up an Oracle server.

The processing model implemented by the Oracle server is that of client-server processing, often referred to as *two-tier*. In the client-server model, the generation of the user interface and much of the application logic is separated from the management of the data. For an application developed using SQL (as all relational database applications will be), this means that the client tier generates the SQL commands, and the server tier executes them. This is the basic client-server split, usually with a local area network dividing the two tiers. The network communications protocol used between the user process and the server process is Oracle's proprietary protocol, Oracle Net.

The client tier consists of two components: the users and the user processes. The server tier has three components: the server processes that execute the SQL, the instance, and the database itself. Each user interacts with a user process. Each user process interacts with a server process, usually across a local area network. The server processes interact with the instance, and the instance with the database. Figure 1-1 shows this relationship diagrammatically. A *session* is a user process in communication with a server process. There will usually be one user process per user and one server process per user process. The user and server processes that make up sessions are launched on demand by users and terminated when no longer required; this is the log-on and log-off cycle. The instance processes and memory structures are launched by the database administrator and persist until the administrator deliberately terminates them; this is the database startup and shutdown cycle.



**Figure 1-1** The indirect connection between a user and a database

The user process can be any client-side software that is capable of connecting to an Oracle server process. Throughout this book, two user processes will be used extensively: SQL\*Plus and SQL Developer. These are simple processes provided by Oracle for establishing sessions against an Oracle server and issuing ad hoc SQL. What the user process actually is does not matter to the Oracle server at all. When an end user fills in a form and clicks a `SUBMIT` button, the user process will generate an `INSERT` statement (detailed in Chapter 8) and send it to a server process for execution against the instance and the database. As far as the server is concerned, the `INSERT` statement might just as well have been typed into SQL\*Plus as what is known as ad hoc SQL.

Never forget that all communication with an Oracle server follows this client-server model. The separation of user code from server code dates back to the earliest releases of the database and is unavoidable. Even if the user process is running on the same machine as the server (as is the case if, for example, one is running a database on one's own laptop PC for development or training purposes), the client-server split is still enforced, and network protocols are still used for the communications between the two processes. Applications running in an application server environment (described in the next section) also follow the client-server model for their database access.

## Application Server

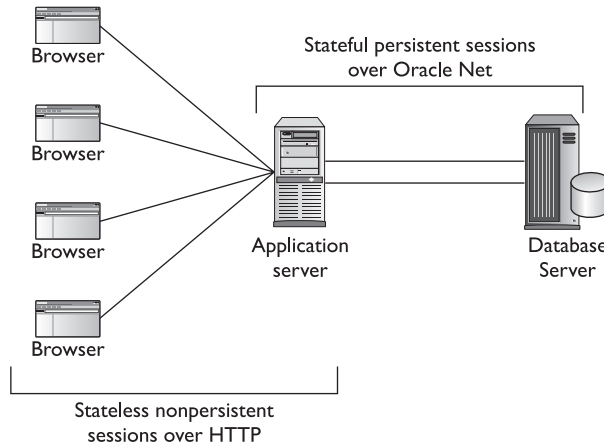
With the emergence of the Web as the de facto standard platform for delivering applications to end users has arisen the need for application servers. An application server allows client-side software, traditionally installed on end-user computers, to be replaced by applications hosted and executing from a centralized location. The application user interface is commonly exposed to users via their web browsers. These applications may make use of data stored in one or more database servers.

Oracle Application Server provides a platform for developing, deploying, and managing *web applications*. A web application can be defined as any application with which users communicate via HTTP. Web applications usually run in at least three tiers: a database tier manages access to the data, the client tier (often implemented via a web browser) handles the local window management for communications with the users, and an application tier in the middle executes the program logic that generates the user interface and the SQL calls to the database.

It is possible for an application to use a one-to-one mapping of end-user session to database session: each user will establish a browser-based session against the application server, and the application server will then establish a session against the database server on the user's behalf. However, this model has been proven to be highly inefficient when compared to the *connection pooling* model. With connection pooling, the application server establishes a relatively small number of persistent database sessions and makes them available on demand (queuing requests if necessary) to a relatively large number of end-user sessions against the application server. Figure 1-2 illustrates the three-tier architecture using connection pooling.

From the point of view of the database, it makes no difference whether a SQL statement comes from a client-side process such as SQL\*Plus or Microsoft Access or from a pooled session to an application server. In the former case, the user process occurs on one machine; in the latter, the user process has been divided into two tiers: an applications tier that generates the user interface and a client tier that displays it.

**Figure 1-2**  
The connection  
pooling model



**TIP** DBAs often find themselves pressed into service as Application Server administrators. Be prepared for this. There is a separate OCP curriculum for Application Server, for which it may well be worth studying.

## Enterprise Manager

The increasing size and complexity of IT installations can make management of each component quite challenging. Management tools can make the task easier, and consequently increase staff productivity.

Oracle Enterprise Manager comes in three forms:

- Database Control
- Application Server Control
- Grid Control

Oracle Enterprise Manager Database Control is a graphical tool for managing one database, which may be a Real Application Clusters (RAC) clustered database. RAC databases are covered in more advanced books; they are mentioned here because they can be managed through the tool. Database Control has facilities for real-time management and monitoring, for running scheduled jobs such as backup operations, and for reporting alert conditions interactively and through e-mail. A RAC database will have a Database Control process running on each node where there is a database instance; these processes communicate with each other, so that each has a complete picture of the state of the RAC.

Oracle Enterprise Manager Application Server Control is a graphical tool for managing one or more application server instances. The technology for managing multiple instances is dependent on the version. Up to and including Oracle Application Server 10g release 2, multiple application servers were managed as a *farm*, with a metadata repository (typically residing in an Oracle database) as the central management point. This is an excellent management model and offers superb capabilities for deploying and maintaining applications, but it is proprietary to Oracle. From Application Server 10g release 3 onward, the technology is based on J2EE clustering, which is not proprietary to Oracle.

Both Database Control and Application Server Control consist of a Java process running on the server machine, which listens for HTTP or HTTPS connection requests. Administrators connect to these processes from a browser. Database Control then connects to the local database server, and Application Server Control connects to the local application server.

Oracle Enterprise Manager Grid Control globalizes the management environment. A management repository (residing in an Oracle database) and one or more management servers manage the complete environment: all the databases and application servers, wherever they may be. Grid Control can also manage the nodes, or machines, on which the servers run, and (through plug-ins) a wide range of third-party products. Each managed node runs an agent process, which is responsible for monitoring the managed targets on the node: executing jobs against them and reporting status, activity levels, and alert conditions back to the management server(s).

Grid Control provides a holistic view of the environment and, if well configured, can significantly enhance the productivity of administration staff. It becomes possible for one administrator to manage effectively hundreds or thousands of targets. The inherent management concept is management by exception. Instead of logging on to each target server to check for errors or problems, Grid Control provides a summary graphic indicating the availability of targets in an environment. The interface supports honing into the targets that are generating exceptions, using drill-down web links, thereby assisting with expedient problem identification.



**EXAM TIP** Anything that can be done with OEM can also be done through SQL statements. The OCP examinations test the use of SQL for administration work extensively. It is vital to be familiar with command-line techniques.

## Oracle Development Tools

Oracle provides several tools for developing applications and utility programs, and supports a variety of languages. The programming languages that are parsed and executed internally within the Oracle Server are Structured Query Language (SQL), Procedural SQL (PL/SQL), and Java. Oracle development technologies written externally to the database include products found in Oracle Developer Suite (Forms, Reports, and Discoverer), Oracle Application Server, and other third-generation languages (3GLs). There is also a wide variety of third-party tools and environments that can be used for developing applications that will connect to an Oracle database; in particular .NET from Microsoft, for which Oracle provides a comprehensive developers' toolkit.

## Internal Languages

SQL is used for data-related activities but cannot be used on its own for developing complete applications. It has no real facilities for developing user interfaces, and it also lacks the procedural structures needed for advanced data manipulation. The other two languages available within the database fill these gaps. They are PL/SQL and Java. PL/SQL is a 3GL proprietary to Oracle. It supports the regular procedural

constructs (such as conditional branching based on if-then-else and iterative looping) and facilities for user interface design. SQL calls may be embedded in the PL/SQL code. Thus, a PL/SQL application might use SQL to retrieve one or more rows from the database, perform various actions based on their content, and then issue more SQL to write rows back to the database. Java offers a similar capability to embed SQL calls within the Java code. This is industry-standard technology: any Java programmer should be able to write code that will work with an Oracle database (or indeed with any other Java-compliant database).

All Oracle DBAs must be fully acquainted with SQL and PL/SQL. This is assumed, and required, knowledge.

Knowledge of Java is not assumed and indeed is rarely required. A main reason for this is that bespoke Java applications are now rarely run within the database. Early releases of Oracle's application server could not run some of the industry-standard Java application components, such as servlets and Enterprise JavaBeans (EJBs). To get around this serious divergence from standards, Oracle implemented a Java engine within the database that did conform to the standards. However, from Oracle Application Server release 9i, it has been possible to run servlets and EJBs where they should be run: on the application server middle tier. Because of this, it has become less common to run Java within the database.

The DBA is likely to spend a large amount of time tuning and debugging SQL and PL/SQL. Oracle's model for the division of responsibility here is clear: the database administrator identifies code with problems and passes it to the developers for fixing. But in many cases, developers lack the skills (or perhaps the inclination) to do this and the database administrator has to fill this role.



**TIP** All DBAs must be fully acquainted with SQL and with PL/SQL. Knowledge of Java and other languages is not usually required but is often helpful.

## External Languages

Other languages are available for developing client-server applications that run externally to the database. The most commonly used are C and Java, but it is possible to use most of the mainstream 3GLs. For most languages, Oracle provides the OCI (Oracle Call Interface) libraries that let code written in these languages connect to an Oracle database and invoke SQL commands.

Applications written in C or other procedural languages make use of the OCI libraries to establish sessions against the database server. These libraries are proprietary to Oracle. This means that any code using them will be specifically written for Oracle, and would have to be substantially rewritten before it could run against any other database. Java applications can avoid this problem. Oracle provides database connectivity for both *thick* and *thin* Java clients.

A thick Java client is Oracle aware. It uses the supplied OCI class library to connect to the database. This means that the application can make use of all the database's capabilities, including features that are unique to the Oracle environment. Java thick-client applications can exploit the database to the full. But they can never work with a third-party product, and they require the OCI client software to be installed.

A thin Java client is not aware of the database against which it is running: it works with a virtual database defined according to the Java standard, and it lets the container within which it is running map this virtual database onto the Oracle database. This gives the application portability across database versions and providers: a thin Java client application could be deployed in a non-Oracle environment without any changes. But any Oracle features that are not part of the Java database connectivity (JDBC) standard will not be available.

The choice between thick and thin Java clients should be made by a team of informed individuals and influenced by a number of factors, including performance; the need for Oracle-specific features; corporate standards; application portability; and programmer productivity. Oracle's JDeveloper tool can be used to develop both thick- and thin-client Java applications.

### **Oracle Developer Suite**

Some organizations do not want to use a 3GL to develop database applications. Oracle Corporation provides rapid application development tools as part of the Oracle Developer Suite. Like the languages, these application development tools end up doing the same thing: constructing SQL statements that are sent to the database server for execution.

Oracle Forms Developer builds applications that run on an Oracle Application Server middle tier and display in a browser. Forms applications are generally quick to develop and are optimized for interfacing with database objects. Specialized triggers and components support feature-rich web-based database applications.

Oracle Reports is a tool for generating and formatting reports, either on demand or according to a schedule. Completed reports can be cached for distribution. Oracle Reports, like Forms, is a full development environment and requires a programmer to generate specialized reports. The huge advantage provided by Oracle Reports is that the output is infinitely customizable and end users can get exactly what they requested.

Oracle Discoverer is a tool for ad hoc report generation that empowers end users to develop reports themselves. Once Oracle Discoverer, which runs on an Oracle Application Server middle tier, has been appropriately configured, programmer input is not needed, since the end users do their own development.

### **Oracle Applications**

The number of Oracle Applications products has increased substantially in recent years due to a large number of corporate acquisitions, but two remain predominant. The Oracle E-Business Suite is a comprehensive suite of applications based around an accounting engine, and Oracle Collaboration Suite is a set of office automation tools.

Oracle E-Business Suite, based around a core set of financial applications, includes facilities for accounting; human resources; manufacturing; customer relationship management; customer services; and much more. All the components share a common data model. The current release has a user interface written with Oracle Developer Forms and Java; it runs on Oracle Application Server and stores data in an Oracle database.

Oracle Collaboration Suite includes (among other things) servers for e-mail, diary management, voicemail and fax, web conferencing, and (perhaps most impressive) file serving. There is complete integration between the various components. The applications run on Oracle Application Servers, and can be accessed through a web interface from browsers or made available on mobile wireless devices.

**Exercise 1-1: Investigate DBMSs in Your Environment** This is a paper-based exercise, with no specific solution.

Identify the applications, application servers, and databases used in your environment. Then, concentrating on the databases, try to get a feeling for how big and busy they are. Consider the number of users; the volatility of the data; the data volumes. Finally, consider how critical they are to the organization: how much downtime or data loss can be tolerated for each applications and database? Is it possible to put a financial figure on this?

The result of this exercise should indicate the criticality of the DBA's role.

## Prerequisite Concepts

The Oracle Database Server product may be installed on a wide variety of hardware platforms and operating systems. Most companies prefer one of the popular Unix operating systems or Microsoft Windows. Increasingly, information technology graduates who opt to pursue a career in the world of Oracle Server technologies lack the exposure to Unix, and you are strongly advised (if you are in such a position), to consider courses on Unix fundamentals, shell scripting, and system administration. In smaller organizations, a DBA may very well concurrently fulfill the roles of system administrator and database administrator (and sometimes even, software developer). As organizations grow in size, IT departments become very segmented and specialized and it is common to have separate Operating Systems, Security, Development, and DBA departments. In fact, larger organizations often have DBA Teams working only with specific operating systems.

This section discusses several basic concepts you need to know to get up and running with an installation of the Oracle database. The actual installation is covered in Chapter 2.

## Oracle Concepts

The Oracle Database Server comprises two primary components called the Instance and the Database. It is easy to get confused since the term "Database" is often used synonymously with the term "Server." The instance component refers to a set of operating system processes and memory structures initialized upon startup, while the database component refers to the physical files used for data storage and database operation. You must therefore expect your Oracle Server installation to consume memory, process, and disk resources on your server. Oracle supplies many tools you may use when interacting with the database, the most common of which are: Oracle Universal Installer (OUI), which is used to install and remove Oracle software;

Database Configuration Assistant (DBCA), which may be used to create, modify, or delete databases; and SQL\*Plus and SQL Developer, which provide interfaces for writing and executing SQL. These tools are described in Chapter 2.

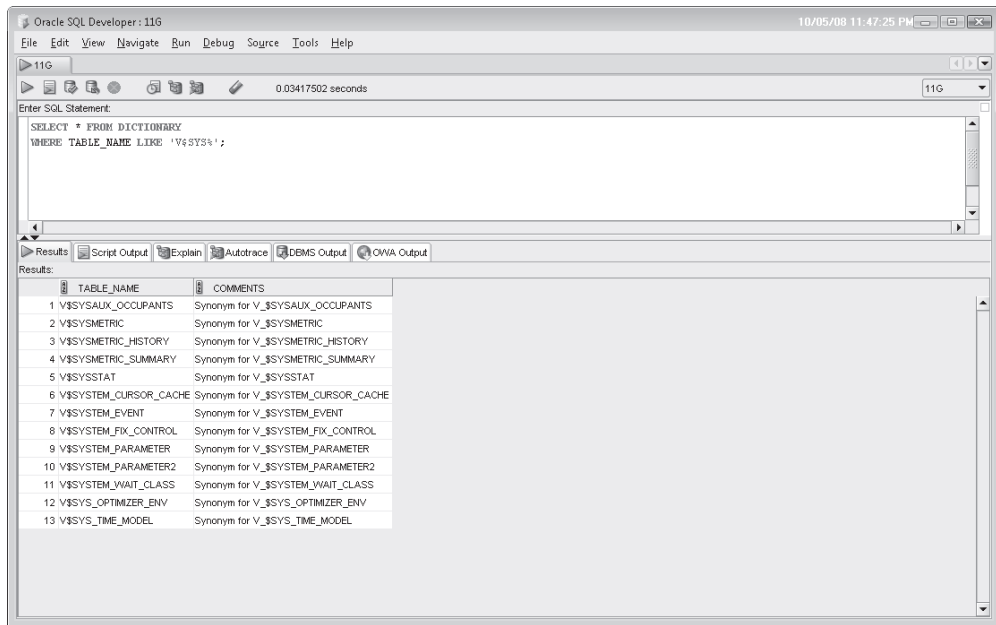
## SQL Concepts

SQL is a powerful language integral to working with Oracle databases. We introduce the concepts of tables, rows, columns, and basic SQL queries here to support your learning as you perform the basic DBA tasks. A complete and thorough discussion of these concepts is detailed in Part 2 of this guide.

### Tables, Rows, and Columns

Data in an Oracle database is primarily stored in two-dimensional relational tables. Each table consists of rows containing data that is segmented across each column. A table may contain many rows but has a fixed number of columns. Data about the Oracle Server itself is stored in a special set of tables known as *data dictionary* tables. Figure 1-3 shows the DICTONARY table comprising two columns called TABLE\_NAME and COMMENTS. Thirteen rows of data have been retrieved from this table.

Relational tables conform to certain rules that constrain and define the data. At the column level, each column must be of a certain data type, such as numeric, date-time, or character. The character data type is the most generic and allows the storage of any character data. At the row level, each row usually has some uniquely identifying characteristic: this could be the value of one column, such as the TABLE\_NAME shown in the example just given, that cannot be repeated in different rows.



The screenshot shows the Oracle SQL Developer interface. The SQL statement entered is: `SELECT * FROM DICTONARY WHERE TABLE_NAME LIKE 'V$SYS%';`. The results are displayed in a table with two columns: TABLE\_NAME and COMMENTS. The results are as follows:

TABLE_NAME	COMMENTS
V\$SYS_AUX_OCCUPANTS	Synonym for V_\$SYS_AUX_OCCUPANTS
V\$SYSMETRIC	Synonym for V_\$SYSMETRIC
V\$SYSMETRIC_HISTORY	Synonym for V_\$SYSMETRIC_HISTORY
V\$SYSMETRIC_SUMMARY	Synonym for V_\$SYSMETRIC_SUMMARY
V\$SYSSTAT	Synonym for V_\$SYSSTAT
V\$SYSTEM_CURSOR_CACHE	Synonym for V_\$SYSTEM_CURSOR_CACHE
V\$SYSTEM_EVENT	Synonym for V_\$SYSTEM_EVENT
V\$SYSTEM_FIX_CONTROL	Synonym for V_\$SYSTEM_FIX_CONTROL
V\$SYSTEM_PARAMETER	Synonym for V_\$SYSTEM_PARAMETER
V\$SYSTEM_PARAMETER2	Synonym for V_\$SYSTEM_PARAMETER2
V\$SYSTEM_WAIT_CLASS	Synonym for V_\$SYSTEM_WAIT_CLASS
V\$SYS_OPTIMIZER_ENV	Synonym for V_\$SYS_OPTIMIZER_ENV
V\$SYS_TIME_MODEL	Synonym for V_\$SYS_TIME_MODEL

Figure 1-3 Querying the DICTONARY table

## Basic Queries

Figure 1-3 introduces a classic SQL query executed using the SQL Developer tool supplied by Oracle. There are many tools that provide a SQL interface to the database, the most common of which is SQL\*Plus. Although the details of SQL queries are discussed in Part 2, they are generally intuitive, and for your immediate needs it is sufficient to interpret the query in Figure 1-3 as follows. The keywords in the statement are SELECT, FROM, WHERE, and LIKE. The asterisk in the first line instructs Oracle to retrieve all columns from the table called DICTONARY. Therefore both columns, called TABLE\_NAME and COMMENTS respectively, are retrieved. The second line contains a conditional WHERE clause that restricts the rows retrieved to only those which have a data value beginning with the characters “V\$SYS” in the TABLE\_NAME column.

## Operating System Concepts

The database installation will consume physical disk storage, and you are encouraged to start considering the hardware you have earmarked for your installation. The two primary disk space consumers are Oracle program files and Oracle database *datafiles*. The program files are often referred to as the Oracle *binaries*, since they collectively represent the compiled C programs essential for creating and maintaining databases. Once the Oracle 11g *binaries* are installed, they consume about 3GB of disk space, but this usage remains relatively stable. The datafiles, however, host the actual rows of data and shrink and grow as the database is used. The default seed database that is relatively empty consumes about 2GB of disk space. Another important hardware consideration is memory (RAM). You will require a minimum of 512MB of RAM, but at least 1GB is required for a usable system.

Most Unix platforms require preinstallation tasks, which involve ensuring that operating system users, groups, patches, kernel parameters, and swap space are adequately specified. Consult with an operating system specialist if you are unfamiliar with these tasks. The superuser (or root) privilege is required to modify these operating system parameters. Commands for checking these resources are described in Chapter 2.

## Single-Instance Architecture

In this book, you will deal largely with the most common database environment: one instance on one computer, opening a database stored on local disks. The more complex distributed architectures, involving multiple instances and multiple databases, are beyond the scope of the OCP examination (though not the OCM qualification), but you may realistically expect to see several high-level summary questions on distributed architectures.

## Single-Instance Database Architecture

The instance consists of memory structures and processes. Its existence is transient, in your RAM and on your CPU(s). When you shut down the running instance, all trace of its existence goes away at the same time. The database consists of physical files, on

disk. Whether running or stopped, these remain. Thus the lifetime of the instance is only as long as it exists in memory: it can be started and stopped. By contrast, the database, once created, persists indefinitely—until you deliberately delete the files that are associated with the database.

The processes that make up the instance are known as *background* processes because they are present and running at all times while the instance is active. These processes are for the most part completely self-administering, though in some cases it is possible for the DBA to influence the number of them and their operation.

The memory structures, which are implemented in shared memory segments provided by the operating system, are known as the *system global area*, or SGA. This is allocated at instance startup and released on shutdown. Within certain limits, the SGA in the 11g instance and the components within it can be resized while the instance is running, either automatically or in response to the DBA's instructions.

User sessions consist of a user process running locally to the user machine connecting to a server process running locally to the instance on the server machine. The technique for launching the server processes, which are started on demand for each session, is covered in Chapter 4. The connection between user process and server process is usually across a local area network and uses Oracle's proprietary Oracle Net protocol layered on top of an industry-standard protocol (usually TCP). The user process-to-server process split implements the client-server architecture: user processes generate SQL; server processes execute SQL. The server processes are sometimes referred to as *foreground* processes, in contrast with the background processes that make up the instance. Associated with each server process is an area of nonsharable memory, called the *program global area*, or PGA. This is private to the session, unlike the system global area, which is available to all the foreground and background processes. Note that background processes also have a PGA. The size of any one session's PGA will vary according to the memory needs of the session at any one time; the DBA can define an upper limit for the total of all the PGAs, and Oracle manages the allocation of this to sessions dynamically.



**TIP** You will sometimes hear the term *shadow process*. Be cautious of using this. Some people use it to refer to foreground processes; others use it for background processes.

Memory management in 11g can be completely automated: the DBA need do nothing more than specify an overall memory allocation for both the SGA and the PGA and let Oracle manage this memory as it thinks best. Alternatively, the DBA can determine memory allocations. There is an in-between technique, where the DBA defines certain limits on what the automatic management can do.



**EXAM TIP** SGA memory is shared across all background and foreground processes; PGA memory can be accessed only by the foreground process of the session to which it has been allocated. Both SGA and PGA memory can be automatically managed.

The physical structures that make up an Oracle database are the datafiles, the redo log, and the controlfile. Within the visible physical structure of the datafiles lie the logical structures seen by end users (developers, business analysts, data warehouse architects, and so on). The Oracle architecture guarantees abstraction of the logical from the physical: there is no need for a programmer to know the physical location of any data, since they only interact with logical structures, such as tables. Similarly, it is impossible for a system administrator to know what data resides in any physical structure: the operating system files, not their contents, are all that is visible. It is only you, the database administrator, who is permitted (and required) to see both sides of the story.

Data is stored in *datafiles*. There is no practical limit to the number or size of datafiles, and the abstraction of logical storage from physical storage means that datafiles can be moved or resized and more datafiles can be added without end users being aware of this. The relationship between physical and logical structures is maintained and documented in the data dictionary, which contains metadata describing the whole database. By querying certain views in the data dictionary, the DBA can determine precisely where every part of every table is located.

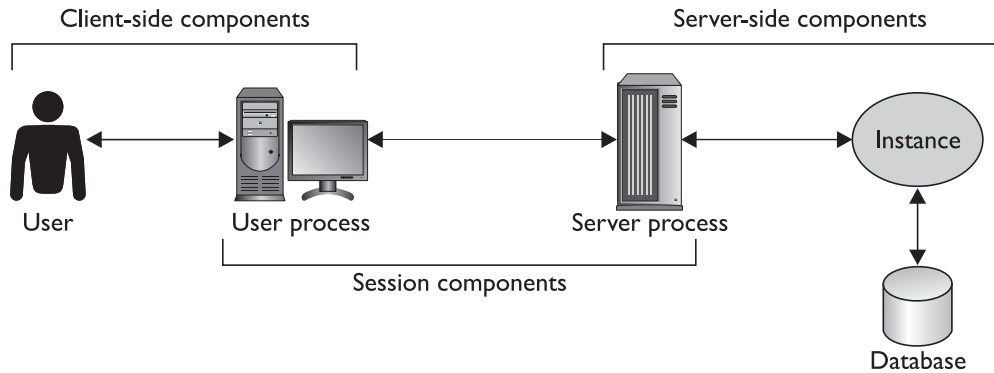
The *data dictionary* is a set of tables stored within the database. There is a recursive problem here: the instance needs to be aware of the physical and logical structure of the database, but the information describing this is itself within the database. The solution to this problem lies in the staged startup process, which is detailed in Chapter 3.

A requirement of the RDBMS standard is that the database must not lose data. This means that it must be backed up, and furthermore that any changes made to data between backups must be captured in such a manner that they can be applied to a restored backup. This is the forward recovery process. Oracle implements the capture of changes through the *redo log*. The redo log is a sequential record of all *change vectors* applied to data. A change vector is the alteration made by a DML (Data Manipulation Language: INSERT, UPDATE, or DELETE) statement. Whenever a user session makes any changes, the data itself in the data block is changed, and the change vector is written out to the redo log, in a form that makes it repeatable. Then in the event of damage to a datafile, a backup of the file can be restored and Oracle will extract the relevant change vectors from the redo log and apply them to the data blocks within the file. This ensures that work will never be lost—unless the damage to the database is so extensive as to lose not only one or more datafiles, but also either their backups or the redo log.

The *controlfile* stores the details of the physical structures of the database and is the starting point for the link to the logical structures. When an instance opens a database, it begins by reading the controlfile. Within the controlfile is information the instance can then use to connect to the rest of the database, and the data dictionary within it.

The architecture of a single-instance database can be summarized as consisting of four interacting components:

- A user interacts with a user process.
- A user process interacts with a server process.
- A server process interacts with an instance.
- An instance interacts with a database.



**Figure 1-4** The indirect connection between a user and a database

Figure 1-4 represents this graphically.

It is absolutely impossible for any client-side process to have any contact with the database: all access must be mediated by server-side processes. The client-server split is between the user process, which generates SQL, and the server process, which executes it.

## Distributed Systems Architectures

In the single-instance environment, one instance opens one database. In a distributed environment, there are various possibilities for grouping instances and databases.

Principally:

- Real Application Clusters (RAC), where multiple instances open one database
- Streams, where multiple Oracle servers propagate transactions between each other
- Dataguard, where a primary database updates a standby database

Combinations of these options can result in a system that can achieve the goals of 100 percent uptime and no data loss, with limitless scalability and performance.

## Real Application Clusters (RAC)

RAC provides amazing capabilities for performance, fault tolerance, and scalability (and possibly cost savings) and is integral to the Oracle's concept of the Grid. With previous releases, RAC (or its precursor, Oracle Parallel Server) was an expensive add-on option, but from database release 10g onward, RAC is bundled with the Standard Edition license. This is an indication of how much Oracle Corporation wants to push users toward the RAC environment. Standard Edition RAC is limited to a certain number of computers and a certain number of CPUs and cores per computer, but

even within these limitations it gives access to a phenomenally powerful environment. RAC is an extra-cost option for the Enterprise Edition, where the scalability becomes effectively limitless: bounded only by the clustering capacity of the underlying operating system and hardware.

A RAC database can be configured for 100 percent uptime. One instance can be brought down (either for planned maintenance, or perhaps because the computer on which it is running crashes) and the database will remain accessible through a surviving instance on another machine. Sessions against the failed instance can be reestablished against a surviving instance without the end user being aware of any disruption.

Transparent scalability comes from the ability to add instances, running on different machines, to a RAC dynamically. They will automatically take on some of the workload without users needing to be aware of the fact that now more instances are available.

Some applications will have a performance benefit from running on a RAC. Parallel processing can improve the performance of some work, such as long-running queries and large batch updates. In a single-instance database, assigning multiple parallel execution servers to such jobs will help—but they will all be running in one instance on one machine. In a RAC database, the parallel execution servers can run on different instances, which may get around some of the bottlenecks inherent in single-instance architecture. Other work, such as processing the large number of small transactions typically found in an OLTP system, will not gain a performance benefit.



---

**TIP** Don't convert to RAC just because you can. You need to be certain of what you want to achieve before embarking on what is a major exercise that may not be necessary.

## Streams

There are various circumstances that make it desirable to transfer data from one database to another. Fault tolerance is one: if an organization has two (or more) geographically separated databases, both containing identical data and both available at all times for users to work on, then no matter what goes wrong at one site, work should be able to continue uninterrupted at the other. Another reason is tuning: the two databases can be configured for different types of work, such as a transaction processing database and a data warehouse.

Keeping the databases synchronized will have to be completely automatic, and all changes made at either site will need to be propagated in real or near-real time to the other site. Another reason could be maintenance of a data warehouse. Data sets maintained by an OLTP database will need to be propagated to the warehouse database, and subsequently these copies will need to be periodically refreshed with changes. The data might then be pushed further out, perhaps to a series of data marts, each with a subset of the warehouse. Streams is a facility for capturing changes made to tables and applying them to remote copies of the tables.

Streams can be bidirectional: identical tables at two or more sites, with all user transactions executed at each site broadcast to and applied at the other sites. This is the streaming model needed for fault tolerance. An alternative model is used in the data warehouse example, where data sets (and ensuing changes made to them) are extracted from tables in one database and pushed out to tables in another database. In this model, the flow of information is more likely to be unidirectional, and the table structures may well not be identical at the downstream sites.

## Dataguard

Dataguard systems have one primary database against which transactions are executed, and one or more standby databases used for fault tolerance or for query processing. The standbys are instantiated from a backup of the primary, and updated (possibly in real time) with all changes applied to the primary.

Standbys come in two forms. A *physical* standby is byte-for-byte identical with the primary, for the purpose of zero data loss. Even if the primary is totally destroyed, all data will be available on the standby. The change vectors applied to the primary are propagated to the physical standby in the form of redo records, and applied as though a restored backup database were being recovered. A *logical* standby contains the same data as the primary, but with possibly different data structures, typically to facilitate query processing. The primary database may have data structures (typically indexes) optimized for transaction processing, while the logical standby may have structures optimized for data warehouse type work. Change vectors that keep the logical standby in synch with the primary are propagated in the form of SQL statements, using the Streams mechanism.

**Exercise 1-2: Determine if the Database Is Single Instance or Part of a Distributed System** In this exercise, you will run queries to determine whether the database is a self-contained system, or if it is part of a larger distributed environment. Either SQL Developer or SQL\*Plus may be used. If you do not have access to an Oracle database yet to practice this exercise, you can skip to Chapter 2, complete an installation, and return to this exercise.

1. Connect to the database as user SYSTEM.
2. Determine if the instance is part of a RAC database:

```
select parallel from v$instance;
```

This will return NO if it is a single-instance database.

3. Determine if the database is protected against data loss by a standby database:

```
select protection_level from v$database;
```

This will return UNPROTECTED if the database is indeed unprotected.

4. Determine if Streams has been configured in the database:

```
select * from dba_streams_administrator;
```

This will return no rows, if Streams has never been configured.

## Instance Memory Structures

An Oracle instance consists of a block of shared memory known as the system global area, or SGA, and a number of background processes. The SGA contains three mandatory data structures:

- The database buffer cache
- The log buffer
- The shared pool

It may, optionally, also contain

- A large pool
- A Java pool
- A Streams pool

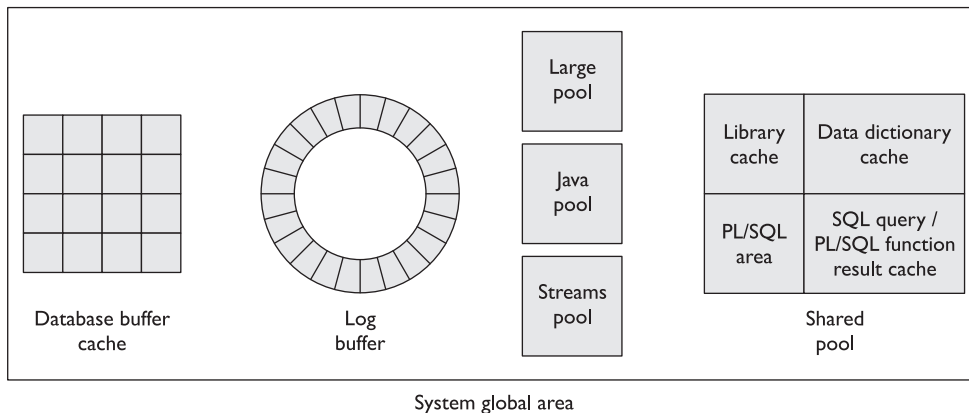
These memory structures are depicted in Figure 1-5, and the three primary structures are detailed in the sections that follow.

User sessions also need memory on the server side. This is nonsharable and is known as the program global area, or PGA. Each session will have its own, private PGA.

Managing the size of these structures can be largely automatic, or the DBA can control the sizing himself. It is generally good practice to use the automatic management.



**EXAM TIP** Which SGA structures are required, and which are optional? The database buffer cache, log buffer, and shared pool are required; the large pool, Java pool, and Streams pool are optional.



**Figure 1-5** The key memory structures present in the SGA

## The Database Buffer Cache

The *database buffer cache* is Oracle's work area for executing SQL. When updating data, users' sessions don't directly update the data on disk. The data blocks containing the data of interest are first copied into the database buffer cache (if they are not already there). Changes (such as inserting new rows and deleting or modifying existing rows) are applied to these copies of the data blocks in the database buffer cache. The blocks will remain in the cache for some time afterward, until the buffer they are occupying is needed for caching another block.

When querying data, the data also goes via the cache. The session works out which blocks contain the rows of interest and copies them into the database buffer cache (if they are not already there); the relevant rows are then transferred into the session's PGA for further processing. And again, the blocks remain in the database buffer cache for some time afterward.

Take note of the term *block*. Datafiles are formatted into fixed-sized blocks. Table rows, and other data objects such as index keys, are stored in these blocks. The database buffer cache is formatted into memory buffers each sized to hold one block. Unlike blocks, rows are of variable length; the length of a row will depend on the number of columns defined for the table, whether the columns actually have anything in them, and if so, what. Depending on the size of the blocks (which is chosen by the DBA) and the size of the rows (which is dependent on the table design and usage), there may be several rows per block or possibly a row may stretch over several blocks. The structure of a data block will be described in the section "Database Storage Structures" later in this chapter.

Ideally, all the blocks containing data that is frequently accessed will be in the database buffer cache, therefore minimizing the need for disk I/O. As a typical use of the database buffer cache, consider a sales rep in the online store retrieving a customer record and updating it, with these statements:

```
select customer_id, customer_name from customers;
update customers set customer_name='Sid' where customer_id=100;
commit;
```

To execute the SELECT statement submitted by the user process, the session's server process will scan the buffer cache for the data block that contains the relevant row. If it finds it, a buffer cache *hit* has occurred. In this example, assume that a buffer cache *miss* occurred and the server process reads the data block containing the relevant row from a datafile into a buffer, before sending the results to the user process, which formats the data for display to the sales rep.

The user process then submits the UPDATE statement and the COMMIT statement to the server process for execution. Provided that the block with the row is still available in the cache when the UPDATE statement is executed, the row will be updated in the buffer cache. In this example, the buffer cache hit ratio will be 50 percent: two accesses of a block in the cache, but only one read of the block from disk. A well-tuned database buffer cache can result in a cache hit ratio well over 90 percent.

A buffer storing a block whose image in the cache is not the same as the image on disk is often referred to as a *dirty* buffer. A buffer will be *clean* when a block is first copied

into it: at that point, the block image in the buffer is the same as the block image on disk. The buffer will become dirty when the block in it is updated. Eventually, dirty buffers must be written back to the datafiles, at which point the buffer will be clean again. Even after being written to disk, the block remains in memory; it is possible that the buffer will not be overwritten with another block for some time.

Note that there is no correlation between the frequency of updates to a buffer (or the number of COMMITs) and when it gets written back to the datafiles. The write to the datafiles is done by the database writer background process.

The size of the database buffer cache is critical for performance. The cache should be sized adequately for caching all the frequently accessed blocks (whether clean or dirty), but not so large that it caches blocks that are rarely needed. An undersized cache will result in excessive disk activity, as frequently accessed blocks are continually read from disk, used, overwritten by other blocks, and then read from disk again. An oversized cache is not so bad (so long as it is not so large that the operating system has to swap pages of virtual memory in and out of real memory) but can cause problems; for example, startup of an instance is slower if it involves formatting a massive database buffer cache.



---

**TIP** Determining the optimal size of the database buffer cache is application specific and a matter of performance tuning. It is impossible to give anything but the vaguest guidelines without detailed observations, but it is probably true to say that the majority of databases will operate well with a cache sized in hundreds of megabytes up to a few gigabytes. Very few applications will perform well with a cache smaller than this, and not many will need a cache of hundreds of gigabytes.

The database buffer cache is allocated at instance startup time. Prior to release 9i of the database it was not possible to resize the database buffer cache subsequently without restarting the database instance, but from 9i onward it can be resized up or down at any time. This resizing can be either manual or (from release 10g onward) automatic according to workload, if the automatic mechanism has been enabled.



---

**TIP** The size of the database buffer cache can be adjusted dynamically and can be automatically managed.

## The Log Buffer

The *log buffer* is a small, short-term staging area for change vectors before they are written to the redo log on disk. A *change vector* is a modification applied to something; executing DML statements generates change vectors applied to data. The redo log is the database's guarantee that data will never be lost. Whenever a data block is changed, the change vectors applied to the block are written out to the redo log, from which they can be extracted and applied to datafile backups if it is ever necessary to restore a datafile.

Redo is not written directly to the redo log files by session server processes. If it were, the sessions would have to wait for disk I/O operations to complete whenever they executed a DML statement. Instead, sessions write redo to the log buffer, in memory. This is much faster than writing to disk. The log buffer (which may contain change vectors from many sessions, interleaved with each other) is then written out to the redo log files. One write of the log buffer to disk may therefore be a batch of many change vectors from many transactions. Even so, the change vectors in the log buffer are written to disk in very nearly real time—and when a session issues a COMMIT statement, the log buffer write really does happen in real time. The writes are done by the log writer background process, the LGWR.

The log buffer is small (in comparison with other memory structures) because it is a very short-term storage area. Change vectors are inserted into it and are streamed to disk in near real time. There is no need for it to be more than a few megabytes at the most, and indeed making it much bigger than the default value can be seriously bad for performance. The default is determined by the Oracle server and is based on the number of CPUs on the server node.

It is not possible to create a log buffer smaller than the default. If you attempt to, it will be set to the default size anyway. It is possible to create a log buffer larger than the default, but this is often not a good idea. The problem is that when a COMMIT statement is issued, part of the commit processing involves writing the contents of the log buffer to the redo log files on disk. This write occurs in real time, and while it is in progress, the session that issued the COMMIT will hang. Commit processing is a critical part of the Oracle architecture. The guarantee that a committed transaction will never be lost is based on this: the commit-complete message is not returned to the session until the data blocks in the cache have been changed (which means that the transaction has been completed) and the change vectors have been written to the redo log on disk (and therefore the transaction could be recovered if necessary). A large log buffer means that potentially there is more to write when a COMMIT is issued, and therefore it may take a longer time before the commit-complete message can be sent, and the session can resume work.



**TIP** Raising the log buffer size above the default may be necessary for some applications, but as a rule start tuning with the log buffer at its default size.

The log buffer is allocated at instance startup, and it cannot be resized without restarting the instance. It is a circular buffer. As server processes write change vectors to it, the current write address moves around. The log writer process writes the vectors out in batches, and as it does so, the space they occupied becomes available and can be overwritten by more change vectors. It is possible that at times of peak activity, change vectors will be generated faster than the log writer process can write them out. If this happens, all DML activity will cease (for a few milliseconds) while the log writer clears the buffer.

The process of flushing the log buffer to disk is one of the ultimate bottlenecks in the Oracle architecture. You cannot do DML faster than the LGWR can flush the change vectors to the online redo log files.



**TIP** If redo generation is the limiting factor in a database's performance, the only option is to go to RAC. In a RAC database, each instance has its own log buffer, and its own LGWR. This is the only way to parallelize writing redo data to disk.



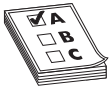
**EXAM TIP** The size of the log buffer is static, fixed at instance startup. It cannot be automatically managed.

## The Shared Pool

The *shared pool* is the most complex of the SGA structures. It is divided into dozens of substructures, all of which are managed internally by the Oracle server. This discussion of architecture will briefly discuss only four of the shared pool components:

- The library cache
- The data dictionary cache
- The PL/SQL area
- The SQL query and PL/SQL function result cache

Several other shared pool structures are described in later chapters. All the structures within the shared pool are automatically managed. Their size will vary according to the pattern of activity against the instance, within the overall size of the shared pool. The shared pool itself can be resized dynamically, either in response to the DBA's instructions or through being managed automatically.



**EXAM TIP** The shared pool size is dynamic and can be automatically managed.

## The Library Cache

The *library cache* is a memory area for storing recently executed code, in its parsed form. *Parsing* is the conversion of code written by programmers into something executable, and it is a process which Oracle does on demand. By caching parsed code in the shared pool, it can be reused greatly improving performance. Parsing SQL code takes time. Consider a simple SQL statement:

```
select * from products where product_id=100;
```

Before this statement can be executed, the Oracle server has to work out what it means, and how to execute it. To begin with, what is `products`? Is it a table, a synonym, or a view? Does it even exist? Then the `"*"`—what are the columns that make up the `products` table (if it is a table)? Does the user have permission to see the table? Answers to these questions and many others have to be found by querying the data dictionary.



**TIP** The algorithm used to find SQL in the library cache is based on the ASCII values of the characters that make up the statement. The slightest difference (even something as trivial as `SELECT` instead of `select`) means that the statement will not match but will be parsed again.

Having worked out what the statement actually means, the server has to decide how best to execute it. Is there an index on the `product_id` column? If so, would it be quicker to use the index to locate the row, or to scan the whole table? More queries against the data dictionary? It is quite possible for a simple one-line query against a user table to generate dozens of queries against the data dictionary, and for the parsing of a statement to take many times longer than eventually executing it. The purpose of the library cache of the shared pool is to store statements in their parsed form, ready for execution. The first time a statement is issued, it has to be parsed before execution—the second time, it can be executed immediately. In a well-designed application, it is possible that statements may be parsed once and executed millions of times. This saves a huge amount of time.

## The Data Dictionary Cache

The data dictionary cache is sometimes referred to as the row cache. Whichever term you prefer, it stores recently used object definitions: descriptions of tables, indexes, users, and other metadata definitions. Keeping such definitions in memory in the SGA, where they are immediately accessible to all sessions, rather than each session having to read them repeatedly from the data dictionary on disk, enhances parsing performance.

The data dictionary cache stores object definitions so that when statements do have to be parsed, they can be parsed quickly—without having to query the data dictionary. Consider what happens if these statements are issued consecutively:

```
select sum(order_amount) from orders;  
select * from orders where order_no=100;
```

Both statements must be parsed because they are different statements—but parsing the first `SELECT` statement will have loaded the definition of the `orders` table and its columns into the data dictionary cache, so parsing the second statement will be faster than it would otherwise have been, because no data dictionary access will be needed.



**TIP** Shared pool tuning is usually oriented toward making sure that the library cache is the right size. This is because the algorithms Oracle uses to allocate memory in the SGA are designed to favor the dictionary cache, so if the library cache is correct, then the dictionary cache will already be correct.

## The PL/SQL Area

Stored PL/SQL objects are procedures, functions, packaged procedures and functions, object type definitions, and triggers. These are all stored in the data dictionary, as source code and also in their compiled form. When a stored PL/SQL object is invoked by a session, it must be read from the data dictionary. To prevent repeated reading, the objects are then cached in the PL/SQL area of the shared pool.

The first time a PL/SQL object is used, it must be read from the data dictionary tables on disk, but subsequent invocations will be much faster, because the object will already be available in the PL/SQL area of the shared pool.



**TIP** PL/SQL can be issued from user processes, rather than being stored in the data dictionary. This is called *anonymous* PL/SQL. Anonymous PL/SQL cannot be cached and reused but must be compiled dynamically. It will therefore always perform worse than stored PL/SQL. Developers should be encouraged to convert all anonymous PL/SQL into stored PL/SQL.

## The SQL Query and PL/SQL Function Result Cache

The result cache is a release 11g new feature. In many applications, the same query is executed many times, by either the same session or many different sessions. Creating a result cache lets the Oracle server store the results of such queries in memory. The next time the query is issued, rather than running the query the server can retrieve the cached result.

The result cache mechanism is intelligent enough to track whether the tables against which the query was run have been updated. If this has happened, the query results will be invalidated and the next time the query is issued, it will be rerun. There is therefore no danger of ever receiving an out-of-date cached result.

The PL/SQL result cache uses a similar mechanism. When a PL/SQL function is executed, its return value can be cached, ready for the next time the function is executed. If the parameters passed to the function, or the tables that the function queries, are different, the function will be reevaluated; otherwise, the cached value will be returned.

By default, use of the SQL query and PL/SQL function result cache is disabled, but if enabled programmatically, it can often dramatically improve performance. The cache is within the shared pool, and unlike the other memory areas described previously, it does afford the DBA some control, as a maximum size can be specified.

## Sizing the Shared Pool

Sizing the shared pool is critical for performance. It should be large enough to cache all the frequently executed code and frequently needed object definitions (in the library cache and the data dictionary cache) but not so large that it caches statements that have only been executed once. An undersized shared pool cripples performance because server sessions have repeatedly to grab space in it for parsing statements, which are then overwritten by other statements and therefore have to be parsed again when they are reexecuted. An oversized shared pool can impact badly on performance because it takes too long to search it. If the shared pool is less than the optimal size, performance will degrade. But there is a minimum size below which statements will fail.

Memory in the shared pool is allocated according to an LRU (least recently used) algorithm. When the Oracle server needs space in the shared pool, it will overwrite the object that has been unused for the longest time. If the object is later needed again, it will have to be reloaded—possibly displacing another object in the shared pool.



**TIP** Determining the optimal size is a matter for performance tuning, but it is probably safe to say that most databases will need a shared pool of several hundred megabytes. Some applications will need more than a gigabyte, and very few will perform adequately with less than a hundred megabytes.

The shared pool is allocated at instance startup time. Prior to release 9i of the database it was not possible to resize the shared pool subsequently without restarting the database instance, but from 9i onward it can be resized up or down at any time. This resizing can be either manual or (from release 10g onward) automatic according to workload, if the automatic mechanism has been enabled.



**EXAM TIP** The shared pool size is dynamic and can be automatically managed.

## The Large Pool

The large pool is an optional area that, if created, will be used automatically by various processes that would otherwise take memory from the shared pool. One major use of the large pool is by shared server processes, described in Chapter 4 in the section “Use the Oracle Shared Server Architecture.” Parallel execution servers will also use the large pool, if there is one. In the absence of a large pool, these processes will use memory on the shared pool. This can cause contention for the shared pool, which may have negative results. If shared servers or parallel servers are being used, a large pool should always be created. Some I/O processes may also make use of the large pool, such as the processes used by the Recovery Manager when it is backing up to a tape device.

Sizing the large pool is not a matter for performance. If a process needs the large pool of memory, it will fail with an error if that memory is not available. Allocating more memory than is needed will not make statements run faster. Furthermore, if a large pool exists, it will be used: it is not possible for a statement to start off by using the large pool, and then revert to the shared pool if the large pool is too small.

From 9i release 2 onward it is possible to create and to resize a large pool after instance startup. With earlier releases, it had to be defined at startup and was a fixed size. From release 10g onward, creation and sizing of the large pool can be completely automatic.



**EXAM TIP** The large pool size is dynamic and can be automatically managed.

## The Java Pool

The Java pool is only required if your application is going to run Java stored procedures within the database: it is used for the heap space needed to instantiate the Java objects. However, a number of Oracle options are written in Java, so the Java pool is considered

standard nowadays. Note that Java code is not cached in the Java pool: it is cached in the shared pool, in the same way that PL/SQL code is cached.

The optimal size of the Java pool is dependent on the Java application, and how many sessions are running it. Each session will require heap space for its objects. If the Java pool is undersized, performance may degrade due to the need to continually reclaim space. In an EJB (Enterprise JavaBean) application, an object such as a stateless session bean may be instantiated and used, and then remain in memory in case it is needed again: such an object can be reused immediately. But if the Oracle server has had to destroy the bean to make room for another, then it will have to be instantiated next time it is needed. If the Java pool is chronically undersized, then the applications may simply fail.

From 10g onward it is possible to create and to resize a large pool after instance startup; this creation and sizing of the large pool can be completely automatic. With earlier releases, it had to be defined at startup and was a fixed size.



**EXAM TIP** The Java pool size is dynamic and can be automatically managed.

## The Streams Pool

The Streams pool is used by Oracle Streams. This is an advanced tool that is beyond the scope of the OCP examinations or this book, but for completeness a short description follows.

The mechanism used by Streams is to extract change vectors from the redo log and to reconstruct statements that were executed from these—or statements that would have the same net effect. These statements are executed at the remote database. The processes that extract changes from redo and the processes that apply the changes need memory: this memory is the Streams pool. From database release 10g it is possible to create and to resize the Streams pool after instance startup; this creation and sizing can be completely automatic. With earlier releases it had to be defined at startup and was a fixed size.



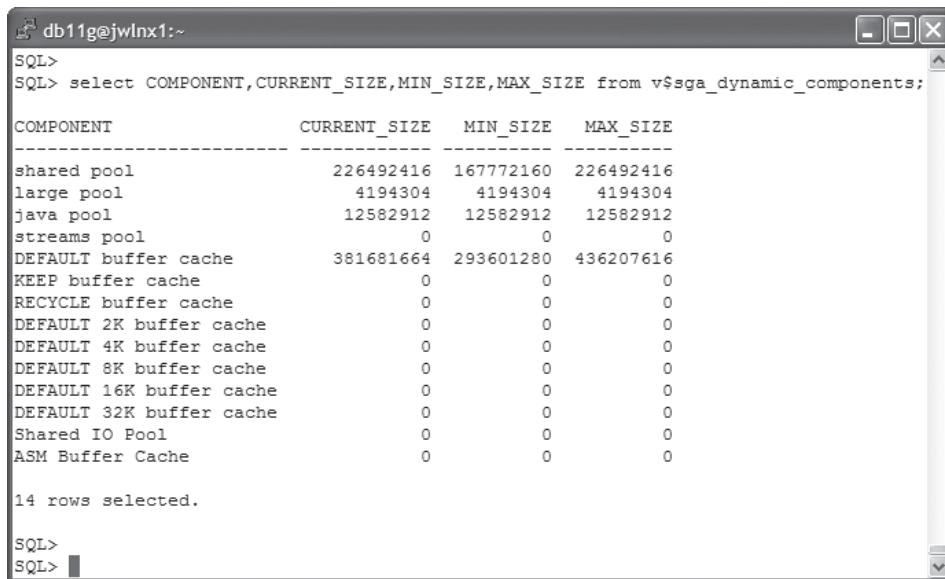
**EXAM TIP** The Streams pool size is dynamic and can be automatically managed.

**Exercise 1-3: Investigate the Memory Structures of the Instance** In this exercise, you will run queries to determine the current sizing of various memory structures that make up the instance. Either SQL Developer or SQL\*Plus may be used.

1. Connect to the database as user SYSTEM.
2. Show the current, maximum, and minimum sizes of the SGA components that can be dynamically resized:

```
select COMPONENT,CURRENT_SIZE,MIN_SIZE,MAX_SIZE
from v$sga_dynamic_components;
```

This illustration shows the result on an example database:



```

db11g@jwlnx1:~
SQL>
SQL> select COMPONENT,CURRENT_SIZE,MIN_SIZE,MAX_SIZE from v$sga_dynamic_components;

COMPONENT                CURRENT_SIZE    MIN_SIZE    MAX_SIZE
-----
shared pool                226492416      167772160  226492416
large pool                 4194304        4194304    4194304
java pool                  12582912       12582912   12582912
streams pool                0              0          0
DEFAULT buffer cache      381681664     293601280  436207616
KEEP buffer cache         0              0          0
RECYCLE buffer cache      0              0          0
DEFAULT 2K buffer cache   0              0          0
DEFAULT 4K buffer cache   0              0          0
DEFAULT 8K buffer cache   0              0          0
DEFAULT 16K buffer cache  0              0          0
DEFAULT 32K buffer cache  0              0          0
Shared IO Pool            0              0          0
ASM Buffer Cache           0              0          0

14 rows selected.

SQL>
SQL>

```

The example shows an instance without Streams, hence a Streams pool of size zero. Neither the large pool nor the Java pool has changed since instance startup, but there have been changes made to the sizes of the shared pool and database buffer cache. Only the default pool of the database buffer cache has been configured; this is usual, except in highly tuned databases.

3. Determine how much memory has been, and is currently, allocated to program global areas:

```

select name,value from v$pgastat
where name in ('maximum PGA allocated','total PGA allocated');

```

## Instance Process Structures

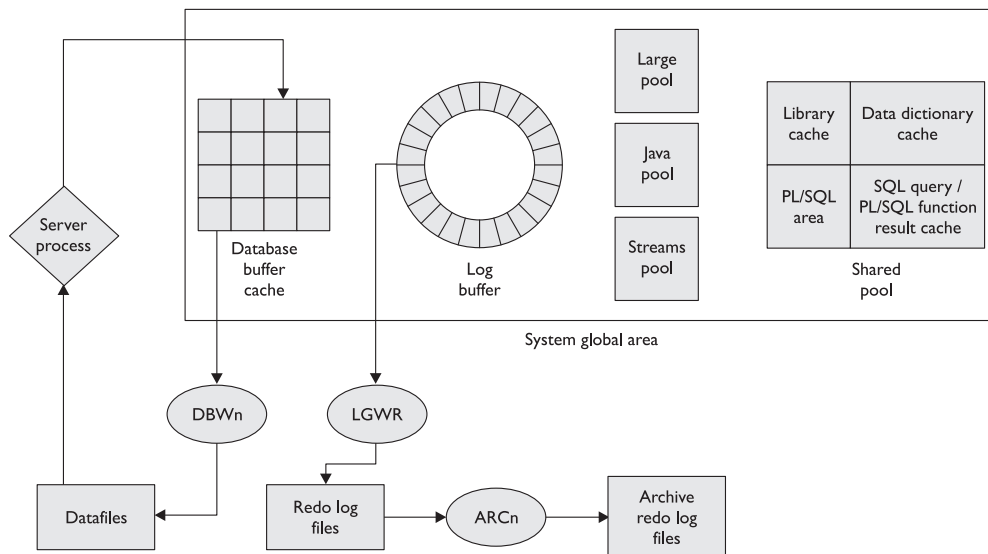
The instance background processes are the processes that are launched when the instance is started and run until it is terminated. There are five background processes that have a long history with Oracle; these are the first five described in the sections that follow: System Monitor (SMON), Process Monitor (PMON), Database Writer (DBWn), Log Writer (LGWR), and Checkpoint Process (CKPT). A number of others have been introduced with the more recent releases; notable among these are Manageability Monitor (MMON) and Memory Manager (MMAN). There are also some that are not essential but will exist in most instances. These include Archiver (ARCn) and Recoverer (RECO). Others will exist only if certain options have been enabled. This last group includes the processes required for RAC and Streams. Additionally, some processes exist that are not properly documented (or are not documented at all). The processes described here are those that every OCP candidate will be expected to know.

Figure 1-6 provides a high-level description of the typical interaction of several key processes and SGA memory structures. The server process is representative of the server side of a client-server connection, with the client component consisting of a user session and user process described earlier. The server process interacts with the datafiles to fetch a data block into the buffer cache. This may be modified by some DML, dirtying the block in the buffer cache. The change vector is copied into the circular log buffer that is flushed in almost real-time by the log writer process (LGWR) to the online redo log files. If archivelog mode of the database is configured, the archiver process (ARCn) copies the online redo log files to an archive location. Eventually, some condition may cause the database writer process (DBWn) to write the dirty block to one of the datafiles. The mechanics of the background processes and their interaction with various SGA structures are detailed in the sections that follow.

There is a platform variation that must be cleared up before discussing processes. On Linux and Unix, all the Oracle processes are separate operating system processes, each with a unique process number. On Windows, there is one operating system process (called ORACLE.EXE) for the whole instance: the Oracle processes run as separate threads within this one process.

## SMON, the System Monitor

SMON initially has the task of mounting and opening a database. The steps involved in this are described in detail in Chapter 3. In brief, SMON *mounts* a database by locating and validating the database controlfile. It then *opens* a database by locating and validating all the datafiles and online log files. Once the database is opened and in use, SMON is responsible for various housekeeping tasks, such as coalescing free space in datafiles.



**Figure 1-6** Typical interaction of instance processes and the SGA

## PMON, the Process Monitor

A user session is a user process that is connected to a server process. The server process is launched when the session is created and destroyed when the session ends. An orderly exit from a session involves the user logging off. When this occurs, any work done will be completed in an orderly fashion, and the server process will be terminated. If the session is terminated in a disorderly manner (perhaps because the user's PC is rebooted), then the session will be left in a state that must be cleared up. PMON monitors all the server processes and detects any problems with the sessions. If a session has terminated abnormally, PMON will destroy the server process, return its PGA memory to the operating system's free memory pool, and roll back any incomplete transaction that may have been in progress.



---

**EXAM TIP** If a session terminates abnormally, what will happen to an active transaction? It will be rolled back, by the PMON background process.

## DBWn, the Database Writer

Always remember that sessions do not as a general rule write to disk. They write data (or changes to existing data) to buffers in the database buffer cache. It is the database writer that subsequently writes the buffers to disk. It is possible for an instance to have several database writers (up to a maximum of twenty), which will be called DBW0, DBW1, and so on: hence the use of the term DBWn to refer to “the” database writer. The default is one database writer per eight CPUs, rounded up.



---

**TIP** How many database writers do you need? The default number may well be correct. Adding more may help performance, but usually you should look at tuning memory first. As a rule, before you optimize disk I/O, ask why there is any need for disk I/O.

DBWn writes dirty buffers from the database buffer cache to the datafiles—but it does not write the buffers as they become dirty. On the contrary: it writes as few buffers as possible. The general idea is that disk I/O is bad for performance, so don't do it unless it really is needed. If a block in a buffer has been written to by a session, there is a reasonable possibility that it will be written to again—by that session, or a different one. Why write the buffer to disk, if it may well be dirtied again in the near future? The algorithm DBWn uses to select dirty buffers for writing to disk (which will clean them) will select only buffers that have not been recently used. So if a buffer is very busy, because sessions are repeatedly reading or writing it, DBWn will not write it to disk. There could be hundreds or thousands of writes to a buffer before DBWn cleans it. It could be that in a buffer cache of a million buffers, a hundred thousand of them are dirty—but DBWn might only write a few hundred of them to disk at a time. These will be the few hundred that no session has been interested in for some time.

DBWn writes according to a very lazy algorithm: as little as possible, as rarely as possible. There are four circumstances that will cause DBWn to write: no free buffers, too many dirty buffers, a three-second timeout, and when there is a checkpoint.



**EXAM TIP** What will cause DBWR to write? No free buffers, too many dirty buffers, a three-second timeout, or a checkpoint.

First, when there are no free buffers. If a server process needs to copy a block into the database buffer cache, it must find a *free buffer*. A free buffer is a buffer that is neither dirty (updated, and not yet written back to disk) nor pinned (a pinned buffer is one that is being used by another session at that very moment). A dirty buffer must not be overwritten because if it were changed, data would be lost, and a pinned buffer cannot be overwritten because the operating system's memory protection mechanisms will not permit this. If a server process takes too long (this length of time is internally determined by Oracle) to find a free buffer, it signals the DBWn to write some dirty buffers to disk. Once this is done, these will be clean, free, and available for use.

Second, there may be too many dirty buffers—"too many" being another internal threshold. No one server process may have had a problem finding a free buffer, but overall, there could be a large number of dirty buffers: this will cause DBWn to write some of them to disk.

Third, there is a three-second timeout: every three seconds, DBWn will clean a few buffers. In practice, this event may not be significant in a production system because the two previously described circumstances will be forcing the writes, but the timeout does mean that even if the system is idle, the database buffer cache will eventually be cleaned.

Fourth, there may be a checkpoint requested. The three reasons already given will cause DBWn to write a limited number of dirty buffers to the datafiles. When a checkpoint occurs, all dirty buffers are written. This could mean hundreds of thousands of them. During a checkpoint, disk I/O rates may hit the roof, CPU usage may go to 100 percent, end user sessions may experience degraded performance, and people may start complaining. Then when the checkpoint is complete (which may take several minutes), performance will return to normal. So why have checkpoints? The short answer is, don't have them unless you have to.



**EXAM TIP** What does DBWn do when a transaction is committed? It does absolutely nothing.

The only moment when a checkpoint is absolutely necessary is as the database is closed and the instance is shut down—a full description of this sequence is given in Chapter 3. A checkpoint writes all dirty buffers to disk: this synchronizes the buffer cache with the datafiles, the instance with the database. During normal operation, the datafiles are always out of date, as they may be missing changes (committed and uncommitted). This does not matter, because the copies of blocks in the buffer cache are up to date, and it is these that the sessions work on. But on shutdown, it is necessary to write everything to disk. Automatic checkpoints only occur on shutdown, but a checkpoint can be forced at any time with this statement:

```
alter system checkpoint;
```

Note that from release 8i onward, checkpoints do not occur on log switch (log switches are discussed in Chapter 14).

The checkpoint described so far is a full checkpoint. Partial checkpoints occur more frequently; they force DBWn to write all the dirty buffers containing blocks from just one or more datafiles rather than the whole database: when a datafile or tablespace is taken offline; when a tablespace is put into backup mode; when a tablespace is made read only. These are less drastic than full checkpoints and occur automatically whenever the relevant event happens.

To conclude, the DBWn writes on a very lazy algorithm: as little as possible, as rarely as possible—except when a checkpoint occurs, when all dirty buffers are written to disk, as fast as possible.

## LGWR, the Log Writer

LGWR writes the contents of the log buffer to the online log files on disk. A write of the log buffer to the online redo log files is often referred to as *flushing* the log buffer.

When a session makes any change (by executing INSERT, UPDATE, or DELETE commands) to blocks in the database buffer cache, before it applies the change to the block it writes out the change vector that it is about to apply to the log buffer. To avoid loss of work, these change vectors must be written to disk with only minimal delay. To this end, the LGWR streams the contents of the log buffer to the online redo log files on disk in very nearly real-time. And when a session issues a COMMIT, the LGWR writes in real-time: the session hangs, while LGWR writes the buffer to disk. Only then is the transaction recorded as committed, and therefore nonreversible.

LGWR is one of the ultimate bottlenecks in the Oracle architecture. It is impossible to perform DML faster than LGWR can write the change vectors to disk. There are three circumstances that will cause LGWR to flush the log buffer: if a session issues a COMMIT; if the log buffer is one-third full; if DBWn is about to write dirty buffers.

First, the write-on-commit. To process a COMMIT, the server process inserts a commit record into the log buffer. It will then hang, while LGWR flushes the log buffer to disk. Only when this write has completed is a commit-complete message returned to the session, and the server process can then continue working. This is the guarantee that transactions will never be lost: every change vector for a committed transaction will be available in the redo log on disk and can therefore be applied to datafile backups. Thus, if the database is ever damaged, it can be restored from backup and all work done since the backup was made can be redone.



**TIP** It is in fact possible to prevent the LGWR write-on-commit. If this is done, sessions will not have to wait for LGWR when they commit: they issue the command and then carry on working. This will improve performance but also means that work can be lost. It becomes possible for a session to COMMIT, then for the instance to crash before LGWR has saved the change vectors. Enable this with caution! It is dangerous, and hardly ever necessary. There are only a few applications where performance is more important than data loss.

Second, when the log buffer is one-third full, LGWR will flush it to disk. This is done primarily for performance reasons. If the log buffer is small (as it usually should be) this one-third-full trigger will force LGWR to write the buffer to disk in very nearly real time even if no one is committing transactions. The log buffer for many applications will be optimally sized at only a few megabytes. The application will generate enough redo to fill one third of this in a fraction of a second, so LGWR will be forced to stream the change vectors to disk continuously, in very nearly real time. Then, when a session does COMMIT, there will be hardly anything to write: so the COMMIT will complete almost instantaneously.

Third, when DBWn needs to write dirty buffers from the database buffer cache to the datafiles, before it does so it will signal LGWR to flush the log buffer to the online redo log files. This is to ensure that it will always be possible to reverse an uncommitted transaction. The mechanism of transaction rollback is fully explained in Chapter 8. For now, it is necessary to know that it is entirely possible for DBWn to write an uncommitted transaction to the datafiles. This is fine, so long as the undo data needed to reverse the transaction is guaranteed to be available. Generating undo data also generates change vectors. As these will be in the redo log files before the datafiles are updated, the undo data needed to roll back a transaction (should this be necessary) can be reconstructed if necessary.

Note that it can be said that there is a three-second timeout that causes LGWR to write. In fact, the timeout is on DBWR—but because LGWR will always write just before DBWn, in effect there is a three-second timeout on LGWR as well.



**EXAM TIP** When will LGWR flush the log buffer to disk? On COMMIT; when the buffer is one-third full; just before DBWn writes.

## CKPT, the Checkpoint Process

The purpose of the CKPT changed dramatically between release 8 and release 8i of the Oracle database. In release 8 and earlier, checkpoints were necessary at regular intervals to make sure that in the event of an instance failure (for example, if the server machine should be rebooted) the database could be *recovered* quickly. These checkpoints were initiated by CKPT. The process of recovery is repairing the damage done by an instance failure; it is fully described in Chapter 14.

After a crash, all change vectors referring to dirty buffers (buffers that had not been written to disk by DBWn at the time of the failure) must be extracted from the redo log, and applied to the data blocks. This is the recovery process. Frequent checkpoints would ensure that dirty buffers were written to disk quickly, thus minimizing the amount of redo that would have to be applied after a crash and therefore minimizing the time taken to recover the database. CKPT was responsible for signaling regular checkpoints.

From release 8i onward, the checkpoint mechanism changed. Rather than letting DBWn get a long way behind and then signaling a checkpoint (which forces DBWn to catch up and get right up to date, with a dip in performance while this is going on)

from 8i onward the DBWn performs *incremental checkpoints* instead of full checkpoints. The incremental checkpoint mechanism instructs DBWn to write out dirty buffers at a constant rate, so that there is always a predictable gap between DBWn (which writes blocks on a lazy algorithm) and LGWR (which writes change vectors in near real time). Incremental checkpointing results in much smoother performance and more predictable recovery times than the older full checkpoint mechanism.



**TIP** The faster the incremental checkpoint advances, the quicker recovery will be after a failure. But performance will deteriorate due to the extra disk I/O, as DBWn has to write out dirty buffers more quickly. This is a conflict between minimizing downtime and maximizing performance.

The CKPT no longer has to signal full checkpoints, but it does have to keep track of where in the redo stream the incremental checkpoint position is, and if necessary instruct DBWn to write out some dirty buffers in order to push the checkpoint position forward. The current checkpoint position, also known as the RBA (the redo byte address), is the point in the redo stream at which recovery must begin in the event of an instance crash. CKPT continually updates the controlfile with the current checkpoint position.



**EXAM TIP** When do full checkpoints occur? Only on request, or as part of an orderly database shutdown.

## MMON, the Manageability Monitor

MMON is a process that was introduced with database release 10g and is the enabling process for many of the self-monitoring and self-tuning capabilities of the database.

The database instance gathers a vast number of statistics about activity and performance. These statistics are accumulated in the SGA, and their current values can be interrogated by issuing SQL queries. For performance tuning and also for trend analysis and historical reporting, it is necessary to save these statistics to long-term storage. MMON regularly (by default, every hour) captures statistics from the SGA and writes them to the data dictionary, where they can be stored indefinitely (though by default, they are kept for only eight days).

Every time MMON gathers a set of statistics (known as a *snapshot*), it also launches the Automatic Database Diagnostic Monitor, the ADDM. The ADDM is a tool that analyses database activity using an expert system developed over many years by many DBAs. It observes two snapshots (by default, the current and previous snapshots) and makes observations and recommendations regarding performance. Chapter 5 describes the use of ADDM (and other tools) for performance tuning.



**EXAM TIP** By default, MMON gathers a snapshot and launches the ADDM every hour.

As well as gathering snapshots, MMON continuously monitors the database and the instance to check whether any alerts should be raised. Use of the alert system is covered in the second OCP exam and discussed in Chapter 24. Some alert conditions (such as warnings when limits on storage space are reached) are enabled by default; others can be configured by the DBA.

## MMNL, the Manageability Monitor Light

MMNL is a process that assists the MMON. There are times when MMON's scheduled activity needs to be augmented. For example, MMON flushes statistical information accumulated in the SGA to the database according to an hourly schedule by default. If the memory buffers used to accumulate this information fill before MMON is due to flush them, MMNL will take responsibility for flushing the data.

## MMAN, the Memory Manager

MMAN is a process that was introduced with database release 10g. It enables the automatic management of memory allocations.

Prior to release 9i of the database, memory management in the Oracle environment was far from satisfactory. The PGA memory associated with session server processes was nontransferable: a server process would take memory from the operating system's free memory pool and never return it—even though it might only have been needed for a short time. The SGA memory structures were static: defined at instance startup time, and unchangeable unless the instance was shut down and restarted.

Release 9i changed that: PGAs can grow and shrink, with the server passing out memory to sessions on demand while ensuring that the total PGA memory allocated stays within certain limits. The SGA and the components within it (with the notable exception of the log buffer) can also be resized, within certain limits. Release 10g automated the SGA resizing: MMAN monitors the demand for SGA memory structures and can resize them as necessary.

Release 11g takes memory management a step further: all the DBA need do is set an overall target for memory usage, and MMAN will observe the demand for PGA memory and SGA memory, and allocate memory to sessions and to SGA structures as needed, while keeping the total allocated memory within a limit set by the DBA.



**TIP** The automation of memory management is one of the major technical advances of the later releases, automating a large part of the DBA's job and giving huge benefits in performance and resource utilization.

## ARCn, the Archiver

This is an optional process as far as the database is concerned, but usually a required process by the business. Without one or more ARCn processes (there can be from one to thirty, named ARC0, ARC1, and so on) it is possible to lose data in the event of a failure. The process and purpose of launching ARCn to create archive log files is described in detail in Chapter 14. For now, only a summary is needed.

All change vectors applied to data blocks are written out to the log buffer (by the sessions making the changes) and then to the *online* redo log files (by the LGWR). There are a fixed number of online redo log files of a fixed size. Once they have been filled, LGWR will overwrite them with more redo data. The time that must elapse before this happens is dependent on the size and number of the online redo log files, and the amount of DML activity (and therefore the amount of redo generated) against the database. This means that the online redo log only stores change vectors for recent activity. In order to preserve a complete history of all changes applied to the data, the online log files must be copied as they are filled and before they are reused. The ARCn process is responsible for doing this. Provided that these copies, known as *archive* redo log files, are available, it will always be possible to recover from any damage to the database by restoring datafile backups and applying change vectors to them extracted from all the archive log files generated since the backups were made. Then the final recovery, to bring the backup right up to date, will come by using the most recent change vectors from the online redo log files.



---

**EXAM TIP** LGWR writes the online log files;ARCn reads them. In normal running, no other processes touch them at all.

Most production transactional databases will run in *archive log mode*, meaning that ARCn is started automatically and that LGWR is not permitted to overwrite an online log file until ARCn has successfully archived it to an archive log file.



---

**TIP** The progress of the ARCn processes and the state of the destination(s) to which they are writing must be monitored. If archiving fails, the database will eventually hang. This monitoring can be done through the alert system.

## RECO, the Recoverer Process

A *distributed* transaction involves updates to two or more databases. Distributed transactions are designed by programmers and operate through database links. Consider this example:

```
update orders set order_status=complete where customer_id=1000;
update orders@mirror set order_status=complete where customer_id=1000;
commit;
```

The first update applies to a row in the local database; the second applies to a row in a remote database identified by the database link MIRROR.

The COMMIT command instructs both databases to commit the transaction, which consists of both statements. A full description of commit processing appears in Chapter 8. Distributed transactions require a *two-phase commit*. The commit in each database must be coordinated: if one were to fail and the other were to succeed, the data overall would be in an inconsistent state. A two-phase commit prepares each database by instructing its LGWRs to flush the log buffer to disk (the first phase), and

once this is confirmed, the transaction is flagged as committed everywhere (the second phase). If anything goes wrong anywhere between the two phases, RECO takes action to cancel the commit and roll back the work in all databases.

## Some Other Background Processes

It is unlikely that processes other than those already described will be examined, but for completeness descriptions of the remaining processes usually present in an instance follow. Figure 1-7 shows a query that lists all the processes running in an instance on a Windows system. There are many more processes that may exist, depending on what options have been enabled, but those shown in the figure will be present in most instances.

The processes not described in previous sections are

- **CJQ0, J000** These manage jobs scheduled to run periodically. The job queue Coordinator, CJQn, monitors the job queue and sends jobs to one of several job queue processes, Jnnn, for execution. The job scheduling mechanism is measured in the second OCP examination and covered in Chapter 22.

```

C:\WINDOWS\system32\cmd.exe - sqlplus / as sysdba
SQL> select program from v$process order by program;
PROGRAM
-----
ORACLE.EXE <ARC0>
ORACLE.EXE <ARC1>
ORACLE.EXE <ARC2>
ORACLE.EXE <ARC3>
ORACLE.EXE <CJQ0>
ORACLE.EXE <CKPT>
ORACLE.EXE <D000>
ORACLE.EXE <DBRM>
ORACLE.EXE <DBW0>
ORACLE.EXE <DIA0>
ORACLE.EXE <DIAG>
ORACLE.EXE <FBAR>
ORACLE.EXE <J000>
ORACLE.EXE <LGWR>
ORACLE.EXE <MMAN>
ORACLE.EXE <MMNL>
ORACLE.EXE <MMON>
ORACLE.EXE <PMON>
ORACLE.EXE <PSP0>
ORACLE.EXE <QMNC>
ORACLE.EXE <RECO>
ORACLE.EXE <S000>
ORACLE.EXE <SHAD>
ORACLE.EXE <SMCO>
ORACLE.EXE <SMON>
ORACLE.EXE <UKTM>
ORACLE.EXE <W000>
ORACLE.EXE <q000>
ORACLE.EXE <q001>
PSEUDO
30 rows selected.
SQL>

```

**Figure 1-7** The background processes typically present in a single instance

- **D000** This is a dispatcher process that will send SQL calls to shared server processes, Snnn, if the shared server mechanism has been enabled. This is described in Chapter 4.
- **DBRM** The database resource manager is responsible for setting resource plans and other Resource Manager–related tasks. Using the Resource Manager is measured in the second OCP examination and covered in Chapter 21.
- **DIA0** The diagnosability process zero (only one is used in the current release) is responsible for hang detection and deadlock resolution. Deadlocks, and their resolution, are described in Chapter 8.
- **DIAG** The diagnosability process (not number zero) performs diagnostic dumps and executes oradebug commands (oradebug is a tool for investigating problems within the instance).
- **FBDA** The flashback data archiver process archives the historical rows of tracked tables into flashback data archives. This is a facility for ensuring that it is always possible to query data as it was at a time in the past.
- **PSP0** The process spawner has the job of creating and managing other Oracle processes, and is undocumented.
- **QMNC, Q000** The queue manager coordinator monitors queues in the database and assigns Qnnn processes to enqueue and dequeue messages to and from these queues. Queues can be created by programmers (perhaps as a means for sessions to communicate) and are also used internally. Streams, for example, use queues to store transactions that need to be propagated to remote databases.
- **SHAD** These appear as TNS V1–V3 processes on a Linux system. They are the server processes that support user sessions. In the figure there is only one, dedicated to the one user process that is currently connected: the user who issued the query.
- **SMCO, W000** The space management coordinator process coordinates the execution of various space management–related tasks, such as proactive space allocation and space reclamation. It dynamically spawns slave processes (Wnnn) to implement the task.
- **VKTM** The virtual keeper of time is responsible for keeping track of time and is of particular importance in a clustered environment.

**Exercise 1-4: Investigate the Processes Running in Your Instance** In this exercise you will run queries to see what background processes are running on your instance. Either SQL Developer or SQL\*Plus may be used.

1. Connect to the database as user SYSTEM.
2. Determine what processes are running, and how many of each:

```
select program from v$session order by program;  
select program from v$process order by program;
```

These queries produce similar results: each process must have a session (even the background processes), and each session must have a process. The processes that can occur multiple times will have a numeric suffix, except for the processes supporting user sessions: these will all have the same name.

3. Demonstrate the launching of server processes as sessions are made, by counting the number of server processes (on Linux or any Unix platform) or the number of Oracle threads (on Windows). The technique is different on the two platforms, because on Linux/Unix, the Oracle processes are separate operating system processes, but on Windows they are threads within one operating system process.

A. On Linux, run this command from an operating system prompt:

```
ps -ef|grep oracle|wc -l
```

This will count the number of processes running that have the string `oracle` in their name; this will include all the session server processes (and possibly a few others).

Launch a SQL\*Plus session, and rerun the preceding command. You can use the `host` command to launch an operating shell from within the SQL\*Plus session. Notice that the number of processes has increased. Exit the session, rerun the command, and you will see that the number has dropped down again. The illustration shows this fact:

```

db11g@jwlnx1:~
[db11g@jwlnx1 ~]$
[db11g@jwlnx1 ~]$ ps -ef | grep oracle | wc -l
4
[db11g@jwlnx1 ~]$ sqlplus system/oracle

SQL*Plus: Release 11.1.0.6.0 - Production on Thu Oct 25 09:51:35 2007

Copyright (c) 1982, 2007, Oracle. All rights reserved.

Connected to:
Oracle Database 11g Enterprise Edition Release 11.1.0.6.0 - Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options

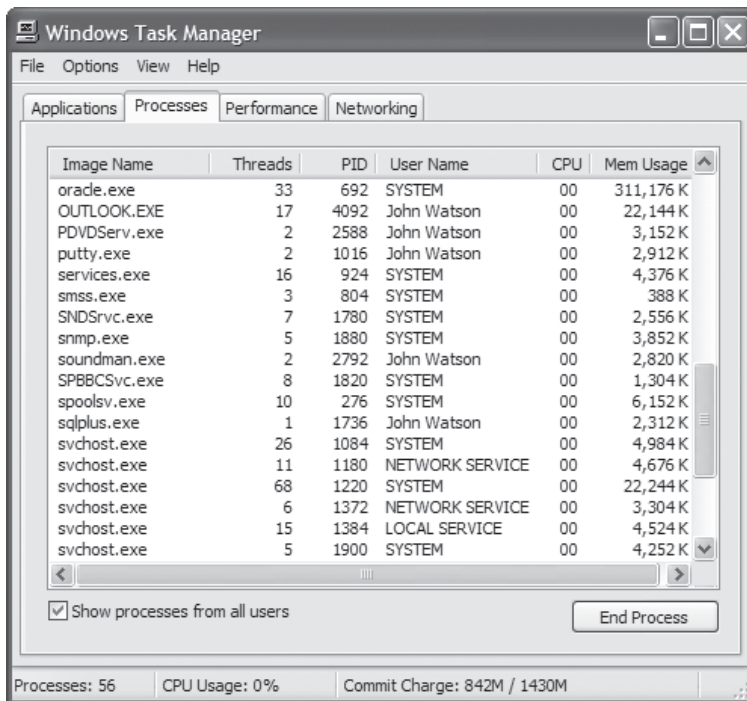
SQL> host
[db11g@jwlnx1 ~]$ ps -ef | grep oracle | wc -l
5
[db11g@jwlnx1 ~]$ exit
exit

SQL> exit
Disconnected from Oracle Database 11g Enterprise Edition Release 11.1.0.6.0 - Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options
[db11g@jwlnx1 ~]$ ps -ef | grep oracle | wc -l
4
[db11g@jwlnx1 ~]$

```

Observe in the illustration how the number of processes changes from 4 to 5 and back again: the difference is the launching and terminating of the server process supporting the SQL\*Plus session.

- B. On Windows, launch the task manager. Configure it to show the number of threads within each process: from the View menu, choose Select Columns and tick the Thread Count check box. Look for the ORACLE.EXE process, and note the number of threads. In the next illustration, this is currently at 33.



Launch a new session against the instance, and you will see the thread count increment. Exit the session, and it will decrement.

## Database Storage Structures

The Oracle database provides complete abstraction of logical storage from physical. The logical data storage is in *segments*. There are various segment types; a typical segment is a table. The segments are stored physically in datafiles. The abstraction of the logical storage from the physical storage is accomplished through tablespaces. The relationships between the logical and physical structures, as well as their definitions, are maintained in the data dictionary.

There is a full treatment of database storage, logical and physical, in Chapter 5.

## The Physical Database Structures

There are three file types that make up an Oracle database, plus a few others that exist externally to the database and are, strictly speaking, optional. The required files are the controlfile, the online redo log files, and the datafiles. The external files that will usually be present (there are others, needed for advanced options) are the initialization parameter file, the password file, the archive redo log files, and the log and trace files.



**EXAM TIP** What three file types must be present in a database? The controlfile, the online redo log files, and any number of datafiles.

### The Controlfile

First a point of terminology: some DBAs will say that a database can have multiple controlfiles, while others will say that it has one controlfile, of which there may be multiple copies. This book will follow the latter terminology, which conforms to Oracle Corporation's use of phrases such as "multiplexing the controlfile," which means to create multiple copies.

The controlfile is small but vital. It contains pointers to the rest of the database: the locations of the online redo log files and of the datafiles, and of the more recent archive log files if the database is in archive log mode. It also stores information required to maintain database integrity: various critical sequence numbers and timestamps, for example. If the Recovery Manager tool (described in Chapters 15, 16, and 17) is being used for backups, then details of these backups will also be stored in the controlfile. The controlfile will usually be no more than a few megabytes big, but your database can't survive without it.

Every database has one controlfile, but a good DBA will always create multiple copies of the controlfile so that if one copy is damaged, the database can quickly be repaired. If all copies of the controlfile are lost, it is possible (though perhaps awkward) to recover, but you should never find yourself in that situation. You don't have to worry about keeping multiplexed copies of the controlfile synchronized—Oracle will take care of that. Its maintenance is automatic—your only control is how many copies to have, and where to put them.

If you get the number of copies, or their location, wrong at database creation time, you can add or remove copies later, or move them around—but you should bear in mind that any such operations will require downtime, so it is a good idea to get it right at the beginning. There is no right or wrong when determining how many copies to have. The minimum is one; the maximum possible is eight. All organizations should have a DBA standards handbook, which will state something like "all production databases will have three copies of the controlfile, on three separate devices," three being a number picked for illustration only, but a number that many organizations are happy with. There is no rule that says two copies is too few, or seven copies is too many; there are only corporate standards, and the DBA's job is to ensure that the databases conform to these.

Damage to any controlfile copy will cause the database instance to terminate immediately. There is no way to avoid this: Oracle Corporation does not permit operating a database with less than the number of controlfiles that have been requested. The techniques for multiplexing or relocating the controlfile are covered in Chapter 14.

## The Online Redo Log Files

The redo log stores a chronologically ordered chain of every change vector applied to the database. This will be the bare minimum of information required to reconstruct, or redo, all work that has been done. If a datafile (or the whole database) is damaged or destroyed, these change vectors can be applied to datafile backups to redo the work, bringing them forward in time until the moment that the damage occurred. The redo log consists of two file types: the online redo log files (which are required for continuous database operation) and the archive log files (which are optional for database operation, but mandatory for point-in-time recovery).

Every database has at least two online redo log files, but as with the controlfile, a good DBA creates multiple copies of each online redo log file. The online redo log consists of groups of online redo log files, each file being known as a *member*. An Oracle database requires at least two groups of at least one member each to function. You may create more than two groups for performance reasons, and more than one member per group for security (an old joke: this isn't just data security, it is job security). The requirement for a minimum of two groups is so that one group can accept the current changes, while the other group is being backed up (or *archived*, to use the correct term).



**EXAM TIP** Every database must have at least two online redo log file groups to function. Each group should have at least two members for safety.

One of the groups is the *current* group: changes are written to the current online redo log file group by LGWR. As user sessions update data in the database buffer cache, they also write out the minimal change vectors to the redo log buffer. LGWR continually flushes this buffer to the files that make up the current online redo log file group. Log files have a predetermined size, and eventually the files making up the current group will fill. LGWR will then perform what is called a log switch. This makes the second group current and starts writing to that. If your database is configured appropriately, the ARCn process(es) will then archive (in effect, back up) the log file members making up the first group. When the second group fills, LGWR will switch back to the first group, making it current, and overwriting it; ARCn will then archive the second group. Thus, the online redo log file groups (and therefore the members making them up) are used in a circular fashion, and each log switch will generate an archive redo log file.

As with the controlfile, if you have multiple members per group (and you should!) you don't have to worry about keeping them synchronized. LGWR will ensure that it writes to all of them, in parallel, thus keeping them identical. If you lose one member of a group, as long as you have a surviving member, the database will continue to function.

The size and number of your log file groups are a matter of tuning. In general, you will choose a size appropriate to the amount of activity you anticipate. The minimum size is fifty megabytes, but some very active databases will need to raise this to several gigabytes if they are not to fill every few minutes. A very busy database can generate megabytes of redo a second, whereas a largely static database may generate only a few megabytes an hour. The number of members per group will be dependent on what level of fault tolerance is deemed appropriate, and is a matter to be documented in corporate standards. However, you don't have to worry about this at database creation time. You can move your online redo log files around, add or drop them, and create ones of different sizes as you please at any time later on. Such operations are performed "online" and don't require downtime—they are therefore transparent to the end users.

## The Datafiles

The third required file type making up a database is the datafile. At a minimum, you must have two datafiles, to be created at database creation time. With previous releases of Oracle, you could create a database with only one datafile—10g and 11g require two, at least one each for the SYSTEM tablespace (that stores the data dictionary) and the SYSAUX tablespace (that stores data that is auxiliary to the data dictionary). You will, however, have many more than that when your database goes live, and will usually create a few more to begin with.

Datafiles are the repository for data. Their size and numbers are effectively unlimited. A small database, of only a few gigabytes, might have just half a dozen datafiles of only a few hundred megabytes each. A larger database could have thousands of datafiles, whose size is limited only by the capabilities of the host operating system and hardware.

The datafiles are the physical structures visible to the system administrators. Logically, they are the repository for the *segments* containing user data that the programmers see, and also for the segments that make up the data dictionary. A segment is a storage structure for data; typical segments are tables and indexes. Datafiles can be renamed, resized, moved, added, or dropped at any time in the lifetime of the database, but remember that some operations on some datafiles may require downtime.

At the operating system level, a datafile consists of a number of operating system blocks. Internally, datafiles are formatted into Oracle *blocks*. These blocks are consecutively numbered within each datafile. The block size is fixed when the datafile is created, and in most circumstances it will be the same throughout the entire database. The block size is a matter for tuning and can range (with limits depending on the platform) from 2KB up to 64KB. There is no relationship between the Oracle block size and the operating system block size.



**TIP** Many DBAs like to match the operating system block size to the Oracle block size. For performance reasons, the operating system blocks should never be larger than the Oracle blocks, but there is no reason not have them smaller. For instance, a 1KB operating system block size and an 8KB Oracle block size is perfectly acceptable.

Within a block, there is a header section and a data area, and possibly some free space. The header section contains information such as the row directory, which lists the location within the data area of the rows in the block (if the block is being used for a table segment) and also row locking information if there is a transaction working on the rows in the block. The data area contains the data itself, such as rows if it is part of a table segment, or index keys if the block is part of an index segment.

When a user session needs to work on data for any purpose, the server process supporting the session locates the relevant block on disk and copies it into a free buffer in the database buffer cache. If the data in the block is then changed (the buffer is dirtied) by executing a DML command against it, eventually DBWn will write the block back to the datafile on disk.



**EXAM TIP** Server processes read from the datafiles; DBWn writes to datafiles.

Datafiles should be backed up regularly. Unlike the controlfile and the online redo log files, they cannot be protected by multiplexing (though they can, of course, be protected by operating system and hardware facilities, such as RAID). If a datafile is damaged, it can be restored from backup and then *recovered* (to recover a datafile means to bring it up to date) by applying all the redo generated since the backup was made. The necessary redo is extracted from the change vectors in the online and archive redo log files. The routines for datafile backup, restore, and recovery are described in Chapters 15–18.

## Other Database Files

These files exist externally to the database. They are, for practical purposes, necessary—but they are not strictly speaking part of the database.

- **The instance parameter file** When an Oracle instance is started, the SGA structures initialize in memory and the background processes start according to settings in the parameter file. This is the only file that needs to exist in order to start an instance. There are several hundred parameters, but only one is required: the `DB_NAME` parameter. All others have defaults. So the parameter file can be quite small, but it must exist. It is sometimes referred to as a `pfile` or `spfile`, and its creation is described in Chapter 3.
- **The password file** Users establish sessions by presenting a username and a password. The Oracle server authenticates these against user definitions stored in the data dictionary. The data dictionary is a set of tables in the database; it is therefore inaccessible if the database is not open. There are occasions when you need to be authenticated before the data dictionary is available: when you need to start the database, or indeed create it. An external password file is one means of doing this. It contains a small number (typically less than half a dozen) of user names and passwords that exist outside the data dictionary, and which can therefore be used to connect to an instance before the data dictionary is available. Creating the password file is described in Chapter 3.

- **Archive redo log files** When an online redo log file fills, the ARCn process copies it to an archive redo log file. Once this is done, the archive log is no longer part of the database in that it is not required for continued operation of the database. It is, however, essential if it is ever necessary to recover a datafile backup, and Oracle does provide facilities for managing the archive redo log files.
- **Alert log and trace files** The alert log is a continuous stream of messages regarding certain critical operations affecting the instance and the database. Not everything is logged: only events that are considered to be really important, such as startup and shutdown; changes to the physical structures of the database; changes to the parameters that control the instance. Trace files are generated by background processes when they detect error conditions, and sometimes to report specific events.

## The Logical Database Structures

The physical structures that make up a database are visible as operating system files to your system administrators. Your users see logical structures such as tables. Oracle uses the term *segment* to describe any structure that contains data. A typical segment is a table, containing rows of data, but there are more than a dozen possible segment types in an Oracle database. Of particular interest (for examination purposes) are table segments, index segments, and undo segments, all of which are investigated in detail later on. For now, you need only know that tables contain rows of information; that indexes are a mechanism for giving fast access to any particular row; and that undo segments are data structures used for storing the information that might be needed to reverse, or roll back, any transactions that you do not wish to make permanent.

Oracle abstracts the logical from the physical storage by means of the *tablespace*. A tablespace is logically a collection of one or more segments, and physically a collection of one or more datafiles. Put in terms of relational analysis, there is a many-to-many relationship between segments and datafiles: one table may be cut across many datafiles, one datafile may contain bits of many tables. By inserting the tablespace entity between the segments and the files, Oracle resolves this many-to-many relationship.

A number of segments must be created at database creation time: these are the segments that make up the data dictionary. These segments are stored in two tablespaces, called SYSTEM and SYSAUX. The SYSAUX tablespace was new with release 10g; in previous releases, the entire data dictionary went into SYSTEM. The database creation process must create at least these two tablespaces, with at least one datafile each, to store the data dictionary.



**EXAM TIP** The SYSAUX tablespace must be created at database creation time in Oracle 10g and later. If you do not specify it, one will be created by default.

A segment consists of a number of blocks. Datafiles are formatted into blocks, and these blocks are assigned to segments as the segments grow. Because managing space

one block at a time would be a time-consuming process, blocks are grouped into *extents*. An extent is a contiguous series of blocks that are consecutively numbered within a datafile, and segments will grow by an extent at a time. These extents need not be adjacent to each other, or even in the same datafile; they can come from any datafile that is part of the tablespace within which the segment resides.

Figure 1-8 shows the Oracle data storage hierarchy, with the separation of logical from physical storage.

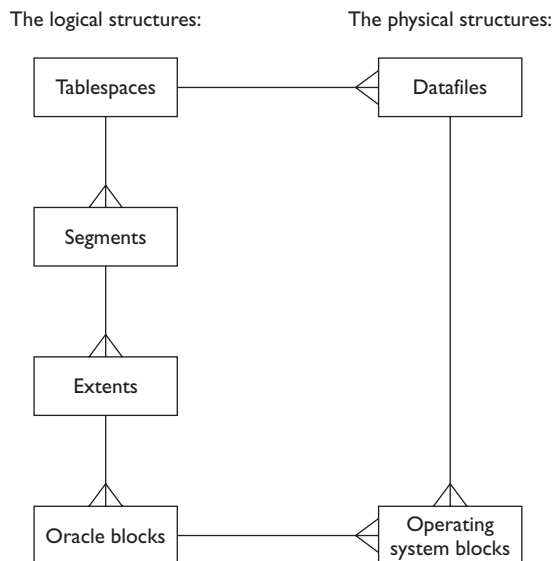
The figure shows the relationships between the storage structures. Logically, a tablespace can contain many segments, each consisting of many extents. An *extent* is a set of Oracle blocks. Physically, a datafile consists of many operating system blocks. The two sides of the model are connected by the relationships showing that one tablespace can consist of multiple datafiles, and at the lowest level that one Oracle block consists of one or more operating system blocks.

## The Data Dictionary

The data dictionary contains metadata that describes the database, both physically and logically, and its contents. User definitions, security information, integrity constraints, and (with release 10g and later) performance monitoring information are all stored in the data dictionary. It is stored as a set of segments in the SYSTEM and SYSAUX tablespaces.

In many ways, the segments that make up the data dictionary are segments like any other: just tables and indexes. The critical difference is that the data dictionary tables are generated at database creation time, and you are not allowed to access them directly. There is nothing to stop an inquisitive DBA from investigating the data dictionary directly, but if you do any updates to it, you may cause irreparable damage to your database, and certainly Oracle Corporation will not support you. Creating a data

**Figure 1-8**  
The Oracle logical  
and physical storage  
hierarchy



dictionary is part of the database creation process. It is maintained subsequently by data definition language commands. When you issue the CREATE TABLE command, you are in fact inserting rows into data dictionary tables, as you are with commands such as CREATE USER or GRANT.

To query the dictionary, Oracle provides a set of views. Most of these views come in three forms, prefixed DBA\_, ALL\_, or USER\_. Any view prefixed USER\_ will describe objects owned by the user querying the view. So no two distinct users will see the same contents while querying a view prefixed with USER\_. If user JOHN queries USER\_TABLES, he will see information about his tables; if you query USER\_TABLES, you will see information about your tables. Any view prefixed ALL\_ will display rows describing objects to which you have access. So ALL\_TABLES shows rows describing your own tables, plus rows describing tables belonging to other users that you have permission to see. Any view prefixed DBA\_ has rows for every object in the database, so DBA\_TABLES has one row for every table in the database, no matter who created it. These views are created as part of the database creation process, along with a large number of PL/SQL packages that are provided by Oracle to assist database administrators in managing the database and programmers in developing applications. PL/SQL code is also stored in the data dictionary.



**EXAM TIP** Which view will show you ALL the tables in the database? DBA\_TABLES, not ALL\_TABLES.

The relationship between tablespaces and datafiles is maintained in the database controlfile. This lists all the datafiles, stating which tablespace they are a part of. Without the controlfile, there is no way that an instance can locate the datafiles and then identify those that make up the SYSTEM tablespace. Only when the SYSTEM tablespace has been opened is it possible for the instance to access the data dictionary, at which point it becomes possible to open the database.

SQL code always refers to objects defined in the data dictionary. To execute a simple query against a table, the Oracle server must first query the data dictionary to find out if the table exists, and the columns that make it up. Then it must find out where, physically, the table is. This requires reading the extent map of the segment. The extent map lists all the extents that make up the table, with the detail of which datafile each extent is in, what block of the datafile the extent starts at, and how many blocks it continues for.

### **Exercise 1-5: Investigate the Storage Structures in Your Database**

In this exercise you will create a table segment, and then work out where it is physically. Either SQL Developer or SQL\*Plus may be used.

1. Connect to the database as user SYSTEM.
2. Create a table without nominating a tablespace—it will be created in your default tablespace, with one extent:

```
create table tab24 (c1 varchar2(10));
```

- Identify the tablespace in which the table resides, the size of the extent, the file number the extent is in, and which block of the file the extent starts at:

```
select tablespace_name, extent_id, bytes, file_id, block_id
from dba_extents where owner='SYSTEM' and segment_name='TAB24';
```

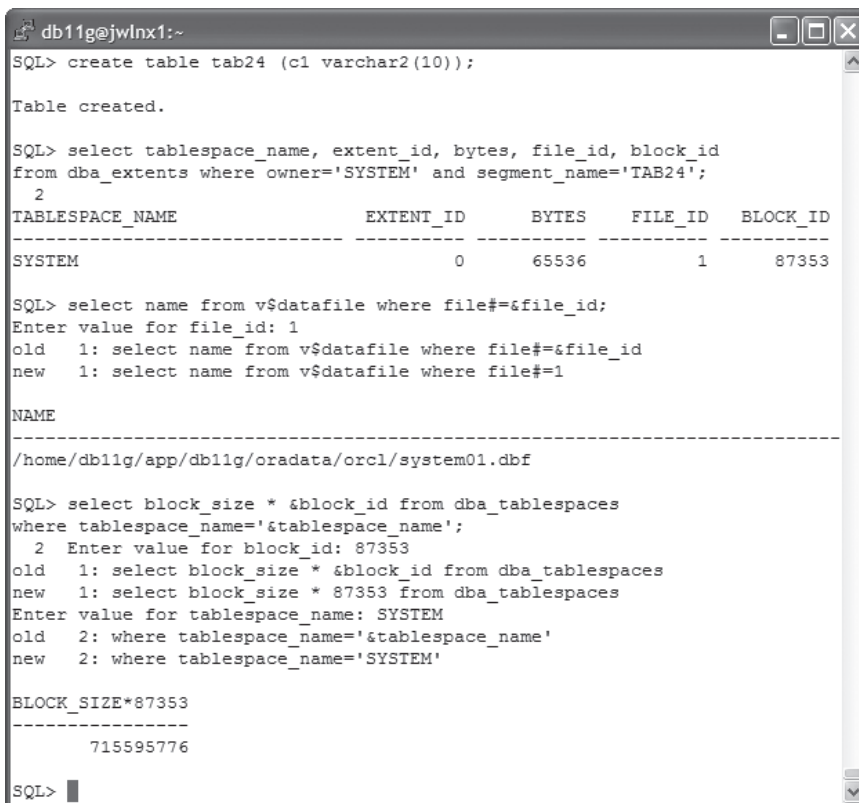
- Identify the file by name: substitute the file\_id from the previous query when prompted:

```
select name from v$datafile where file#=&file_id;
```

- Work out precisely where in the file the extent is, in terms of how many bytes into the file it begins. This requires finding out the tablespace's block size. Enter the block\_id and tablespace\_name returned by the query in Step 3 when prompted.

```
select block_size * &block_id from dba_tablespaces
where tablespace_name='&tablespace_name';
```

The illustration that follows shows these steps, executed from SQL\*Plus:



```
db11g@jwlnx1:~
SQL> create table tab24 (c1 varchar2(10));

Table created.

SQL> select tablespace_name, extent_id, bytes, file_id, block_id
from dba_extents where owner='SYSTEM' and segment_name='TAB24';
 2
TABLESPACE_NAME          EXTENT_ID      BYTES      FILE_ID      BLOCK_ID
-----
SYSTEM                    0             65536       1            87353

SQL> select name from v$datafile where file#=&file_id;
Enter value for file_id: 1
old  1: select name from v$datafile where file#=&file_id
new  1: select name from v$datafile where file#=1

NAME
-----
/home/db11g/app/db11g/oradata/orcl/system01.dbf

SQL> select block_size * &block_id from dba_tablespaces
where tablespace_name='&tablespace_name';
 2 Enter value for block_id: 87353
old  1: select block_size * &block_id from dba_tablespaces
new  1: select block_size * 87353 from dba_tablespaces
Enter value for tablespace_name: SYSTEM
old  2: where tablespace_name='&tablespace_name'
new  2: where tablespace_name='SYSTEM'

BLOCK_SIZE*87353
-----
715595776

SQL>
```

The illustration shows that the table exists in one extent that is 64KB large. This extent is in the file `/home/db11g/app/db11g/oradata/orcl/system01.dbf` and begins about 700MB into the file.

## Two-Minute Drill

### Single-Instance Architecture

- An Oracle server is an instance connected to a database.
- An instance is a block of shared memory and a set of background processes.
- A database is a set of files on disk.
- A user session is a user process connected to a server process.

### Instance Memory Structures

- The instance shared memory is the system global area (the SGA).
- A session's private memory is its program global area (the PGA).
- The SGA consists of a number of substructures, some of which are required (the database buffer cache, the log buffer, and the shared pool) and some of which are optional (the large pool, the Java pool, and the Streams pool).
- The SGA structures can be dynamically resized and automatically managed, with the exception of the log buffer.

### Instance Process Structures

- Session server processes are launched on demand when users connect.
- Background processes are launched at instance startup and persist until shutdown.
- Server processes read from the database; background processes write to the database.
- Some background processes will always be present (in particular SMON, PMON, DBWn, LGWR, CKPT, and MMON); others will run depending on what options have been enabled.

### Database Storage Structures

- There are three required file types in a database: the controlfile, the online redo log files, and the datafiles.
- The controlfile stores integrity information and pointers to the rest of the database.
- The online redo logs store recent change vectors applied to the database.
- The datafiles store the data.
- External files include the parameter file, the password file, archive redo logs, and the log and trace files.

- Logical data storage (segments) is abstracted from physical data storage (datafiles) by tablespaces.
- A tablespace can consist of multiple datafiles.
- Segments consist of multiple extents, which consist of multiple Oracle blocks, which consist of one or more operating system blocks.
- A segment can have extents in several datafiles.

## Self Test

1. Which statements regarding instance memory and session memory are correct? (Choose two answers.)
  - A. SGA memory is private memory segments; PGA memory is shared memory segments.
  - B. Sessions can write to the PGA, not the SGA.
  - C. The SGA is written to by all sessions; a PGA is written by one session.
  - D. The PGA is allocated at instance startup.
  - E. The SGA is allocated at instance startup.
2. How do sessions communicate with the database? (Choose the best answer.)
  - A. Server processes use Oracle Net to connect to the instance.
  - B. Background processes use Oracle Net to connect to the database.
  - C. User processes read from the database and write to the instance.
  - D. Server processes execute SQL received from user processes.
3. What memory structures are a required part of the SGA? (Choose three answers.)
  - A. The database buffer cache
  - B. The Java pool
  - C. The large pool
  - D. The log buffer
  - E. The program global area
  - F. The shared pool
  - G. The Streams pool
4. Which SGA memory structure(s) cannot be resized dynamically after instance startup? (Choose one or more correct answers.)
  - A. The database buffer cache
  - B. The Java pool
  - C. The large pool
  - D. The log buffer
  - E. The shared pool

- F. The Streams pool
  - G. All SGA structures can be resized dynamically after instance startup
5. Which SGA memory structure(s) cannot be resized automatically after instance startup? (Choose one or more correct answers.)
- A. The database buffer cache
  - B. The Java pool
  - C. The large pool
  - D. The log buffer
  - E. The shared pool
  - F. The Streams pool
  - G. All SGA structures can be resized automatically after instance startup
6. When a session changes data, where does the change get written? (Choose the best answer.)
- A. To the data block in the cache, and the redo log buffer
  - B. To the data block on disk, and the current online redo log file
  - C. The session writes to the database buffer cache, and the log writer writes to the current online redo log file
  - D. Nothing is written until the change is committed
7. Which of these background processes is optional? (Choose the best answer.)
- A. ARCn, the archive process
  - B. CKPT, the checkpoint process
  - C. DBWn, the database writer
  - D. LGWR, the log writer
  - E. MMON, the manageability monitor
8. What happens when a user issues a COMMIT? (Choose the best answer.)
- A. The CKPT process signals a checkpoint.
  - B. The DBWn process writes the transaction's changed buffers to the datafiles.
  - C. The LGWR flushes the log buffer to the online redo log.
  - D. The ARCn process writes the change vectors to the archive redo log.
9. An Oracle instance can have only one of some processes, but several of others. Which of these processes can occur several times? (Choose three answers.)
- A. The archive process
  - B. The checkpoint process
  - C. The database writer process
  - D. The log writer process
  - E. The session server process

10. How can one segment can be spread across many datafiles? (Choose the best answer.)
  - A. By allocating an extent with blocks in multiple datafiles
  - B. By spreading the segment across multiple tablespaces
  - C. By assigning multiple datafiles to a tablespace
  - D. By using an Oracle block size that is larger than the operating system block size
11. Which statement is correct regarding the online redo log? (Choose the best answer.)
  - A. There must be at least one log file group, with at least one member.
  - B. There must be at least one log file group, with at least two members.
  - C. There must be at least two log file groups, with at least one member each.
  - D. There must be at least two log file groups, with at least two members each.
12. Where is the current redo byte address, also known as the incremental checkpoint position, recorded? (Choose the best answer.)
  - A. In the controlfile
  - B. In the current online log file group
  - C. In the header of each datafile
  - D. In the system global area

## Self Test Answers

1.  C and E. The SGA is shared memory, updated by all sessions; PGAs are private to each session. The SGA is allocated at startup time (but it can be modified later).  
 A, B, and D. A is wrong because it reverses the situation: it is the SGA that exists in shared memory, not the PGA. B is wrong because sessions write to both their own PGA and to the SGA. D is wrong because (unlike the SGA) the PGA is only allocated on demand.
2.  D. This is the client-server split: user processes generate SQL; server processes execute SQL.  
 A, B, and C. A and B are wrong because they get the use of Oracle Net wrong. Oracle Net is the protocol between a user process and a server process. C is wrong because it describes what server processes do, not what user processes do.
3.  A, D, and F. Every instance must have a database buffer cache, a log buffer, and a shared pool.

- B, C, E, and G. B, C, and G are wrong because the Java pool, the large pool, and the Streams pool are only needed for certain options. E is wrong because the PGA is not part of the SGA at all.
4.  D. The log buffer is fixed in size at startup time.
- A, B, C, E, F, and G. A, B, C, E, and F are wrong because these are the SGA's resizable components. G is wrong because the log buffer is static.
5.  D. The log buffer cannot be resized manually, never mind automatically.
- A, B, C, E, F, and G. A, B, C, E, and F are wrong because these SGA components can all be automatically managed. G is wrong because the log buffer is static.
6.  A. The session updates the copy of the block in memory and writes out the change vector to the log buffer.
- B, C, and D. B is wrong, because while this will happen, it does not happen when the change is made. C is wrong because it confuses the session making changes in memory with LGWR propagating changes to disk. D is wrong because all changes to data occur in memory as they are made—the COMMIT is not relevant.
7.  A. Archiving is not compulsory (though it is usually a good idea).
- B, C, D, and E. CKPT, DBWn, LGWR, and MMON are all necessary processes.
8.  C. On COMMIT, the log writer flushes the log buffer to disk. No other background processes need do anything.
- A, B, and D. A is wrong because checkpoints only occur on request, or on orderly shutdown. B is wrong because the algorithm DBWn uses to select buffers to write to the datafiles is not related to COMMIT processing, but to how busy the buffer is. D is wrong because ARCn only copies filled online redo logs; it doesn't copy change vectors in real time.
9.  A, C, and E. A and C are correct because the DBA can choose to configure multiple archive and database writer processes. E is correct because one server process will be launched for every concurrent session.
- B and D. These are wrong because an instance can have only one log writer process and only one checkpoint process.
10.  C. If a tablespace has several datafiles, segments can have extents in all of them.
- A, B, and D. A is wrong because one extent consists of consecutive block in one datafile. B is wrong because one segment can only exist in one tablespace (though one tablespace can contain many segments). D is wrong because while this can certainly be done, one block can only exist in one datafile.

11.  C. Two groups of one member is the minimum required for the database to function.
- A, B, and D. A and B are wrong because at least two groups are always required. D is wrong because while it is certainly advisable to multiplex the members, it is not a mandatory requirement.
12.  A. The checkpoint process writes the RBA to the controlfile.
- B, C, and D. The online logs, the datafiles, and SGA have no knowledge of where the current RBA is.