Introduction to the Oracle Server

Databases and Information Management

A database server is the key to solving the problems of information management. In general, a server must reliably manage a large amount of data in a multi-user environment so that many users can concurrently access the same data. All this must be accomplished while delivering high performance. A database server must also prevent unauthorized access and provide efficient solutions for failure recovery. The Oracle Server provides efficient and effective solutions with the following features:

client/server (distributed processing) environments

To take full advantage of a given computer system or network, Oracle allows processing to be split between the database server and the client application programs. The computer running the database management system handles all of the database server responsibilities while the workstations running the database application concentrate on the interpretation and display of data.

large databases and space management

Oracle supports the largest of databases, potentially terabytes in size. To make efficient use of expensive hardware devices, it allows full control of space usage.

many concurrent database users

Oracle supports large numbers of concurrent users executing a variety of database applications operating on the same data. It minimizes data contention and guarantees data concurrency.

high transaction processing performance

Oracle maintains the preceding features with a high degree of overall system performance. Database users do not suffer from slow processing performance.

high availability

At some sites, Oracle works 24 hours per day with no down time to limit database throughput. Normal system operations such as database backup and partial computer system failures do not interrupt database use.

controlled availability

Oracle can selectively control the availability of data, at the database level and sub-database level. For example, an administrator can disallow use of a specific application so that the application's data can be reloaded, without affecting other applications.

openness, industry standards

Oracle adheres to industry accepted standards for the data access language, operating systems, user interfaces, and network communication protocols. It is an "open" system that protects a customer's investment.

To protect against unauthorized database access and use, Oracle provides fail-safe security features to limit and monitor data access. These features make it easy to manage even the most complex design for data access.

database enforced integrity

Oracle enforces data integrity, "business rules" that dictate the standards for acceptable data. As a result, the costs of coding and managing checks in many database applications are eliminated.

distributed systems

For networked, distributed environments, Oracle combines the data physically located on different computers into one logical database that can be accessed by all network users. Distributed systems have the same degree of user transparency and data consistency as non-distributed systems, yet receive the advantages of local database management.

Oracle also offers the heterogeneous option that allows users to access data on some non-Oracle databases transparently.

portability
Oracle software is ported to work under different operating systems. Applications developed for Oracle can be ported to any operating system with little or no modification.
compatibility
Oracle software is compatible with industry standards, including most industry standard operating systems. Applications developed for Oracle can be used on virtually any system with little or no modification.
connectibility
Oracle software allows different types of computers and operating systems to share information across networks.
replicated environments
Oracle software lets you replicate groups of tables and their supporting objects to multiple sites. Oracle supports replication of both data- and schema-level changes to these sites. Oracle's flexible replication technology supports basic primary site replication as well as advanced dynamic and shared-ownership models.
The following sections provide a comprehensive overview of the Oracle architecture. Each major section describes a different part of the overall architecture.

The Oracle Server

The Oracle Server is a relational database management system that provides an open, comprehensive, and integrated approach to information management. An Oracle Server consists of an Oracle database and an Oracle instance. The following sections describe the relationship between the database and the instance.

Structured Query Language (SQL)

SQL (pronounced SEQUEL) is the programming language that defines and manipulates the database. SQL databases are relational databases; this means simply that data is stored in a set of simple relations. A database can have one or more tables. And each table has columns and rows. A table that has an employee database, for example, might have a column called employee number and each row in that column would be an employee's employee number.
You can define and manipulate data in a table with SQL commands. You use data definition language (DDL) commands to set up the data. DDL commands include commands to creating and altering databases and tables.
You can update, delete, or retrieve data in a table with data manipulation commands (DML). DML commands include commands to alter and fetch dat. The most common SQL command is the SELECT command, which allows you to retrieve data from the database.
In addition to SQL commands, the Oracle Server has a procedural language called PL/SQL. PL/SQL enables the programmer to program SQL statements. It allows you to control the flow of a SQL program, to use variables, and to write error-handling procedures.

Database Structure

An Oracle database has both a physical and a logical structure. Because the physical and logical server structure are separate, the physical storage of data can be managed without affecting the access to logical storage structures.
Physical Database Structure An Oracle database's physical structure is determined by the operating system files that constitute the database. Each Oracle database is made of three types of files: one or more datafiles, two or more redo log files, and one or more control files. The files of an Oracle database provide the actual physical storage for database information.
Logical Database Structure An Oracle database's logical structure is determined by

- one or more tablespaces. (A tablespace is a logical area of storage explained later in this chapter.)
- the database's schema objects. A *schema* is a collection of objects. *Schema objects* are the logical structures that directly refer to the database's data. Schema objects include such structures as tables, views, sequences, stored procedures, synonyms, indexes, clusters, and database links.

The logical storage structures, including tablespaces, segments, and extents, dictate how the physical space of a database is used. The schema objects and the relationships among them form the relational design of a database.

An Oracle Instance

Every time a database is started, a system global area (SGA) is allocated and Oracle background processes are started. The system global area is a an area of memory used for database information shared by the database users. The combination of the background processes and memory buffers is called an Oracle *instance*. Figure 1 - 1 illustrates a multiple process Oracle instance.

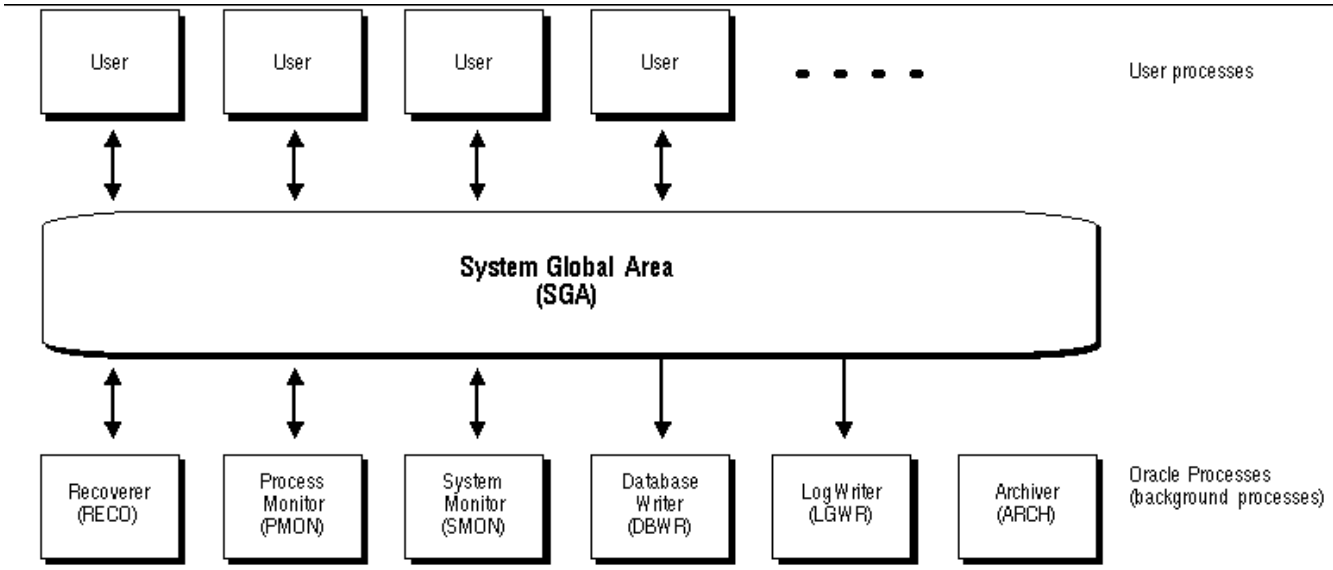

Figure 1 - 1. An Oracle Instance

Ilustración 1

An Oracle instance has two types of processes: user processes and Oracle processes.
A user process executes the code of an application program (such as an Oracle Forms application) or an Oracle Tool (such as Server Manager).
Oracle processes are server processes that perform work for user processes and background processes that perform maintenance work for the Oracle Server.

Communications Software and SQL*Net

If the user and server processes are on different computers of a network or if the user processes connect to shared server processes through dispatcher processes, the user process and server process communicate using SQL*Net.
Dispatchers are optional background processes, present only when a multi-threaded server configuration is used.
*SQL*Net* is Oracle's interface to standard communications protocols that allows for the proper transmission of data between computers.

The Oracle Parallel Server: Multiple Instance Systems

Some hardware architectures (for example, shared disk systems) allow multiple computers to share access to data, software, or peripheral devices. Oracle with the Parallel Server option can take advantage of such architecture by running multiple instances that "share" a single physical database. In appropriate applications, the Oracle Parallel Server allows access to a single database by the users on multiple machines with increased performance.

Database Structure and Space Management

This section describes the architecture of an Oracle database, including the physical and logical structures that make up a database. This discussion provides an understanding of Oracle's solutions to controlled data availability, the separation of logical and physical data structures, and fine-grained control of disk space management.

Relational Database Management Systems

Database management systems have evolved from hierarchical to network to relational models. Today, the most widely accepted database model is the relational model. The relational model has three major aspects:

Structures
Structures are well-defined objects (such as tables, views, indexes, and so on) that store or access the data of a database. Structures and the data contained within them can be manipulated by operations.
Operations
Operations are clearly defined actions that allow users to manipulate the data and structures of a database. The operations on a database must adhere to a predefined set of integrity rules.
Integrity Rules
Integrity rules are the laws that govern which operations are allowed on the data and structures of a database. Integrity rules protect the data and the structures of a database.
Relational database management systems offer benefits such as

- independence of physical data storage and logical database structure
- variable and easy access to all data
- complete flexibility in database design
- reduced data storage and redundancy

An Oracle *database* is a collection of data that is treated as a unit. The general purpose of a database is to store and retrieve related information. The database has *logical structures* and *physical structures*.

Open and Closed Databases

An Oracle database can be *open* (accessible) or *closed* (not accessible). In normal situations, the database is open and available for use. However, the database is sometimes closed for specific administrative functions that require the database's data to be unavailable to users.

Logical Database Structures

The following sections explain logical database structures, including tablespaces, schema objects, data blocks, extents, and segments.

Tablespaces

A database is divided into logical storage units called *tablespaces*. A tablespace is used to group related logical structures together. For example, tablespaces commonly group all of an application's objects to simplify certain administrative operations.
Databases, Tablespaces, and Datafiles The relationship among databases, tablespaces, and datafiles (datafiles are described in the next section) is illustrated in Figure 1 - 2.

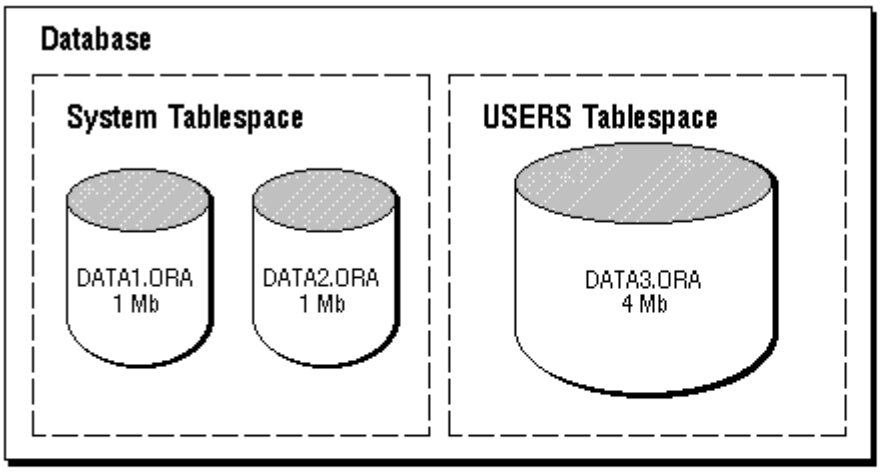

Figure 1 - 2. Databases, Tablespaces, and Datafiles

This figure illustrates the following:

- Each database is logically divided into one or more tablespaces.
- One or more datafiles are explicitly created for each tablespace to physically store the data of all logical structures in a tablespace.
- The combined size of a tablespace's datafiles is the total storage capacity of the tablespace (SYSTEM tablespace has 2 Mb storage capacity while USERS tablespace has 4 Mb).
- The combined storage capacity of a database's tablespaces is the total storage capacity of the database (6 Mb).

Online and Offline Tablespaces A tablespace can be *online* (accessible) or *offline* (not accessible). A tablespace is normally online so that users can access the information within the tablespace. However, sometimes a tablespace may be taken offline to make a portion of the database unavailable while allowing normal access to the remainder of the database. This makes many administrative tasks easier to perform.

Schemas and Schema Objects

A *schema* is a collection of objects. *Schema objects* are the logical structures that directly refer to the database's data. Schema objects include such structures as tables, views, sequences, stored procedures, synonyms, indexes, clusters, and database links. (There is no relationship between a tablespace and a schema; objects in the same schema can be in different tablespaces, and a tablespace can hold objects from different schemas.)

Tables A *table* is the basic unit of data storage in an Oracle database. The tables of a database hold all of the user-accessible data.

Table data is stored in *rows* and *columns*. Every table is defined with a *table name* and set of columns. Each column is given a *column name*, a *datatype* (such as CHAR, DATE, or NUMBER), and a *width* (which may be predetermined by the datatype, as in DATE) or *scale* and *precision* (for the NUMBER datatype only). Once a table is created, valid rows of data can be inserted into it. The table's rows can then be queried, deleted, or updated.

To enforce defined business rules on a table's data, integrity constraints and triggers can also be defined for a table.

Views A *view* is a custom-tailored presentation of the data in one or more tables. A view can also be thought of as a "stored query".

Views do not actually contain or store data; rather, they derive their data from the tables on which they are based, referred to as the *base tables* of the views. Base tables can in turn be tables or can themselves be views.

Like tables, views can be queried, updated, inserted into, and deleted from, with restrictions. All operations performed on a view actually affect the base tables of the view.

Views are often used to do the following:

- Provide an additional level of table security by restricting access to a predetermined set of rows and columns of a table. For example, a view of a table can be created so that columns with sensitive data (for example, salary information) are not included in the definition of the view.
- Hide data complexity. For example, a single view can combine 12 monthly sales tables to provide a year of data for analysis and reporting. A single view can also be used to create a *join*, which is a display of related columns or rows in multiple tables. However, the view hides the fact that this data actually originates from several tables.
- Simplify commands for the user. For example, views allow users to select information from multiple tables without requiring the users to actually know how to perform a correlated subquery.
- Present the data in a different perspective from that of the base table. For example, views provide a means to rename columns without affecting the tables on which the view is based.
- Store complex queries. For example, a query might perform extensive calculations with table information. By saving this query as a view, the calculations are performed only when the view is queried.

Views that involve a join (a SELECT statement that selects data from multiple tables) of two or more tables can only be updated under certain conditions.

Sequences A *sequence* generates a serial list of unique numbers for numeric columns of a database's tables. Sequences simplify application programming by automatically generating unique numerical values for the rows of a single table or multiple tables.

For example, assume two users are simultaneously inserting new employee rows into the EMP table. By using a sequence to generate unique employee numbers for the EMPNO column, neither user has to wait for the other to input the next available employee number. The sequence automatically generates the correct values for each user. Sequence numbers are independent of tables, so the same sequence can be used for one or more tables. After creation, a sequence can be accessed by various users to generate actual sequence numbers.

Program Units The term "program unit" is used in this manual to refer to stored procedures, functions, and packages.

Note: The information in this section applies only to Oracle with the procedural option installed (PL/SQL.).

A *procedure* or *function* is a set of SQL and PL/SQL (Oracle's procedural language extension to SQL) statements grouped together as an executable unit to perform a specific task. For more information about SQL and PL/SQL Procedures and functions allow you to combine the ease and flexibility of SQL with the procedural functionality of a structured programming language. Using PL/SQL, such procedures and functions can be defined and stored in the database for continued use. Procedures and functions are identical, except that functions always return a single value to the caller, while procedures do not return a value to the caller.

Packages provide a method of encapsulating and storing related procedures, functions, and other package constructs together as a unit in the database. While packages provide the database administrator or application developer organizational benefits, they also offer increased functionality and database performance.

Synonyms A *synonym* is an alias for a table, view, sequence, or program unit. A synonym is not actually an object itself, but instead is a direct reference to an object. Synonyms are used to

- mask the real name and owner of an object
- provide public access to an object
- provide location transparency for tables, views, or program units of a remote database
- simplify the SQL statements for database users

A synonym can be public or private. An individual user can create a *private synonym*, which is available only to that user. Database administrators most often create *public synonyms* that make the base schema object available for general, system-wide use by any database user.

Indexes, Clusters, and Hash Clusters Indexes, clusters, and hash clusters are optional structures associated with tables, which can be created to increase the performance of data retrieval.

Indexes are created to increase the performance of data retrieval. Just as the index in this manual helps you locate specific information faster than if there were no index, an Oracle index provides a faster access path to table data. When processing a request, Oracle can use some or all of the available indexes to locate the requested rows efficiently. Indexes are useful when applications often query a table for a range of rows (for example, all employees with a salary greater than 1000 dollars) or a specific row.

Indexes are created on one or more columns of a table. Once created, an index is automatically maintained and used by Oracle. Changes to table data (such as adding new rows, updating rows, or deleting rows) are automatically incorporated into all relevant indexes with complete transparency to the users.

Indexes are logically and physically independent of the data. They can be dropped and created any time with no effect on the tables or other indexes. If an index is dropped, all applications continue to function; however, access to previously indexed data may be slower.

Clusters are an optional method of storing table data. Clusters are groups of one or more tables physically stored together because they share common columns and are often used together. Because related rows are physically stored together, disk access time improves.

The related columns of the tables in a cluster are called the *cluster key*. The cluster key is indexed so that rows of the cluster can be retrieved with a minimum amount of I/O. Because the data in a cluster key of an index cluster (a non-hash cluster) is stored only once for multiple tables, clusters may store a set of tables more efficiently than if the tables were stored individually (not clustered). Figure 1 - 3 illustrates how clustered and non-clustered data is physically stored.

Clusters also can improve performance of data retrieval, depending on data distribution and what SQL operations are most often performed on the clustered data. In particular, clustered tables that are queried in joins benefit from the use of clusters because the rows common to the joined tables are retrieved with the same I/O operation.

Like indexes, clusters do not affect application design. Whether or not a table is part of a cluster is transparent to users and to applications. Data stored in a clustered table is accessed via SQL in the same way as data stored in a non-clustered table.

Hash clusters also cluster table data in a manner similar to normal, index clusters (clusters keyed with an index rather than a hash function). However, a row is stored in a hash cluster based on the result of applying a *hash function* to the row's cluster key value. All rows with the same hash key value are stored together on disk.

Hash clusters are a better choice than using an indexed table or index cluster when a table is often queried with equality queries (for example, return all rows for department 10). For such queries, the specified cluster key value is hashed. The resulting hash key value points directly to the area on disk that stores the specified rows.

Database Links A *database link* is a named object that describes a "path" from one database to another. Database links are implicitly used when a reference is made to a *global object name* in a distributed database.

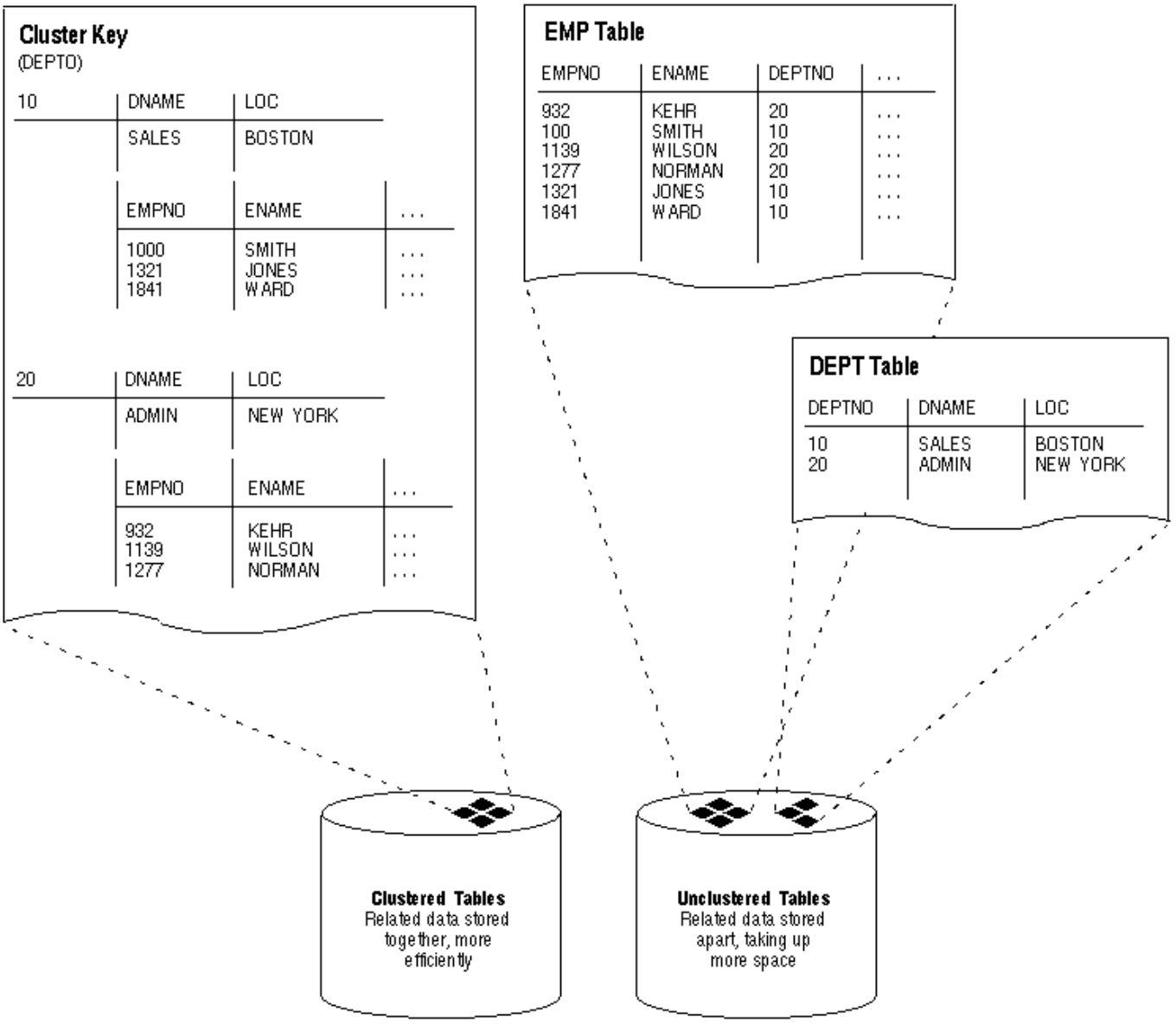

Figure 1 - 3. Clustered and Unclustered Tables

Data Blocks, Extents, and Segments

Oracle allows fine-grained control of disk space usage through the logical storage structures, including data blocks, extents, and segments. For more information on these, see Chapter 3 of this manual.

Oracle Data Blocks At the finest level of granularity, an Oracle database's data is stored in *data blocks*. One data block corresponds to a specific number of bytes of physical database space on disk. A data block size is specified for each Oracle database when the database is created. A database uses and allocates free database space in Oracle data blocks.

Extents The next level of logical database space is called an extent. An *extent* is a specific number of contiguous data blocks, obtained in a single allocation, used to store a specific type of information.

Segments The level of logical database storage above an extent is called a segment. A *segment* is a set of extents allocated for a certain logical structure. For example, the different types of segments include the following:

Data Segment

Each non-clustered table has a data segment. All of the table's data is stored in the extents of its data segment. Each cluster has a data segment. The data of every table in the cluster is stored in the cluster's data segment.

Index Segment

Each index has an index segment that stores all of its data.

Rollback Segment

One or more rollback segments are created by the database administrator for a database to temporarily store "undo" information. This information is used:

- to generate read-consistent database information.
- during database recovery.
- to rollback uncommitted transactions for users

Temporary Segment

Temporary segments are created by Oracle when a SQL statement needs a temporary work area to complete execution. When the statement finishes execution, the temporary segment's extents are returned to the system for future use.

Oracle dynamically allocates space when the existing extents of a segment become full. Therefore, when the existing extents of a segment are full, Oracle allocates another extent for that segment as needed. Because extents are allocated as needed, the extents of a segment may or may not be contiguous on disk.

Physical Database Structures

The following sections explain the physical database structures of an Oracle database, including datafiles, redo log files, and control files.

Datafiles

Every Oracle database has one or more physical *datafiles*. A database's datafiles contain all the database data. The data of logical database structures such as tables and indexes is physically stored in the datafiles allocated for a database.

The following are characteristics of datafiles:

- A datafile can be associated with only one database.
- Database files can have certain characteristics set to allow them to automatically extend when the database runs out of space.
- One or more datafiles form a logical unit of database storage called a tablespace, as discussed earlier in this chapter.

The Use of Datafiles The data in a datafile is read, as needed, during normal database operation and stored in the memory cache of Oracle. For example, assume that a user wants to access some data in a table of a database. If the requested information is not already in the memory cache for the database, it is read from the appropriate datafiles and stored in memory.

Modified or new data is not necessarily written to a datafile immediately. To reduce the amount of disk access and increase performance, data is pooled in memory and written to the appropriate datafiles all at once, as determined by the DBWR background process of Oracle. (For more information about Oracle's memory and process structures and the algorithm for writing database data to the datafiles.

Redo Log Files

Every Oracle database has a set of two or more *redo log files*. The set of redo log files for a database is collectively known as the database's *redo log*. The primary function of the redo log is to record all changes made to data. Should a failure prevent modified data from being permanently written to the datafiles, the changes can be obtained from the redo log and work is never lost.

Redo log files are critical in protecting a database against failures. To protect against a failure involving the redo log itself, Oracle allows a *multiplexed redo log* so that two or more copies of the redo log can be maintained on different disks.

The Use of Redo Log Files The information in a redo log file is used only to recover the database from a system or media failure that prevents database data from being written to a database's datafiles.

For example, if an unexpected power outage abruptly terminates database operation, data in memory cannot be written to the datafiles and the data is lost. However, any lost data can be recovered when the database is opened, after power is restored. By applying the information in the most recent redo log files to the database's datafiles, Oracle restores the database to the time at which the power failure occurred.

The process of applying the redo log during a recovery operation is called *rolling forward*.

Control Files

Every Oracle database has a *control file*. A control file contains entries that specify the physical structure of the database. For example, it contains the following types of information:

- database name
- names and locations of a database's datafiles and redo log files
- time stamp of database creation

Like the redo log, Oracle allows the control file to be multiplexed for protection of the control file.

The Use of Control Files Every time an instance of an Oracle database is started, its control file is used to identify the database and redo log files that must be opened for database operation to proceed. If the physical makeup of the database is altered (for example, a new datafile or redo log file is created), the database's control file is automatically modified by Oracle to reflect the change.

A database's control file is also used if database recovery is necessary.

The Data Dictionary

Each Oracle database has a *data dictionary*. An Oracle data dictionary is a set of tables and views that are used as a *read-only* reference about the database. For example, a data dictionary stores information about both the logical and physical structure of the database. In addition to this valuable information, a data dictionary also stores such information as

- the valid users of an Oracle database
- information about integrity constraints defined for tables in the database
- how much space is allocated for a schema object and how much of it is being used

A data dictionary is created when a database is created. To accurately reflect the status of the database at all times, the data dictionary is automatically updated by Oracle in response to specific actions (such as when the structure of the database is altered). The data dictionary is critical to the operation of the database, which relies on the data dictionary to record, verify, and conduct ongoing work. For example, during database operation, Oracle reads the data dictionary to verify that schema objects exist and that users have proper access to them.

Oracle Server Architecture

The following section discusses the memory and process structures used by an Oracle Server to manage a database. Among other things, the architectural features discussed in this section provide an understanding of Oracle's capabilities to support

- many users concurrently accessing a single database
- the high performance required by concurrent multi-user, multi-application database systems

Memory Structures and Processes

An Oracle Server uses memory structures and processes to manage and access the database. All memory structures exist in the main memory of the computers that constitute the database system. *Processes* are jobs or tasks that work in the memory of these computers.

Figure 1 - 4 shows a typical variation of the Oracle Server memory and process structures.

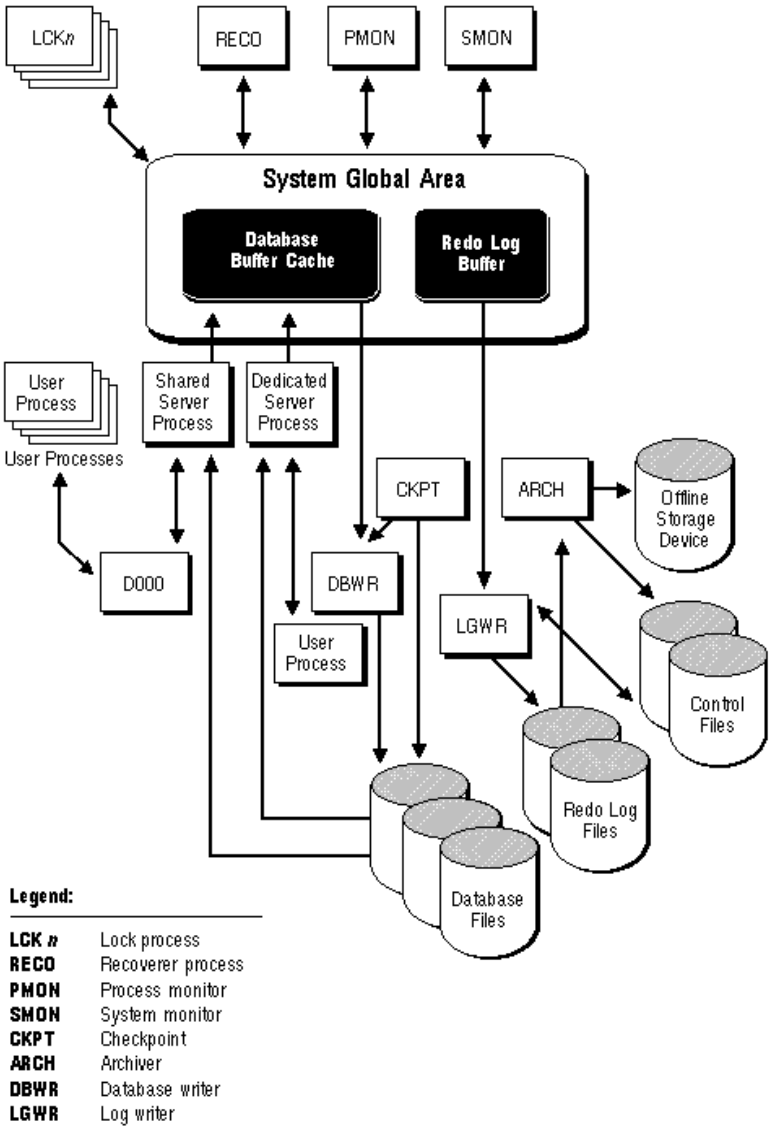

Figure 1 - 4. Memory Structures and Processes of Oracle

Memory Structures

Oracle creates and uses memory structures to complete several jobs. For example, memory is used to store program code being executed and data that is shared among users. Several basic memory structures are associated with Oracle: the system global area (which includes the database buffers, redo log buffers, and the shared pool) and the program global areas. The following sections explain each in detail.

System Global Area (SGA)

The *System Global Area (SGA)* is a shared memory region that contains data and control information for one Oracle instance. An SGA and the Oracle background processes constitute an Oracle instance.
Oracle allocates the system global area when an instance starts and deallocates it when the instance shuts down. Each instance has its own system global area.
Users currently connected to an Oracle Server share the data in the system global area. For optimal performance, the entire system global area should be as large as possible (while still fitting in real memory) to store as much data in memory as possible and minimize disk I/O. The information stored within the system global area is divided into several types of memory structures, including the database buffers, redo log buffer, and the shared pool. These areas have fixed sizes and are created during instance startup.
Database Buffer Cache *Database buffers* of the system global area store the most recently used blocks of database data; the set of database buffers in an instance is the *database buffer cache*. These buffers can contain modified data

that has not yet been permanently written to disk. Because the most recently (and often the most frequently) used data is kept in memory, less disk I/O is necessary and performance is increased.
Redo Log Buffer The *redo log buffer* of the system global area stores *redo entries* -- a log of changes made to the database. The redo entries stored in the redo log buffers are written to an online redo log file, which is used if database recovery is necessary. Its size is static.
Shared Pool The shared pool is a portion of the system global area that contains shared memory constructs such as shared SQL areas. A shared SQL area is required to process every unique SQL statement submitted to a database. A shared SQL area contains information such as the parse tree and execution plan for the corresponding statement. A single shared SQL area is used by multiple applications that issue the same statement, leaving more shared memory for other uses.
Cursors A *cursor* is a handle (a name or pointer) for the memory associated with a specific statement. Although most Oracle users rely on the automatic cursor handling of the Oracle utilities, the programmatic interfaces offer application designers more control over cursors. For example, in precompiler application development, a cursor is a named resource available to a program and can be specifically used for the parsing of SQL statements embedded within the application. The application developer can code an application so that it controls the phases of SQL statement execution and thus improve application performance.

Program Global Area (PGA)

The *Program Global Area (PGA)* is a memory buffer that contains data and control information for a server process. A PGA is created by Oracle when a server process is started. The information in a PGA depends on the configuration of Oracle.

Processes

A *process* is a "thread of control" or a mechanism in an operating system that can execute a series of steps. Some operating systems use the terms *job* or *task*. A process normally has its own private memory area in which it runs. An Oracle Server has two general types of processes: user processes and Oracle processes.

User (Client) Processes

A *user process* is created and maintained to execute the software code of an application program (such as a Pro*C/C++ program) or an Oracle tool (such as Server Manager). The user process also manages the communication with the server processes. User processes communicate with the server processes through the program interface, described later in this section.

Oracle Processes

Oracle processes are called by other processes to perform functions on behalf of the invoking process. The different types of Oracle processes and their specific functions are discussed in the following sections.
Server Processes Oracle creates *server processes* to handle requests from connected user processes. A server process is in charge of communicating with the user process and interacting with Oracle to carry out requests of the associated user process. For example, if a user queries some data that is not already in the database buffers of the system global area, the associated server process reads the proper data blocks from the datafiles into the system global area.
Oracle can be configured to vary the number of user processes per server process. In a *dedicated server configuration*, a server process handles requests for a single user process. A *multi-threaded server configuration* allows many user processes to share a small number of server processes, minimizing the number of server processes and maximizing the utilization of available system resources.
On some systems, the user and server processes are separate, while on others they are combined into a single process. If a system uses the multi-threaded server or if the user and server processes run on different machines, the user and server processes must be separate. Client/server systems separate the user and server processes and execute them on different machines.
Background Processes Oracle creates a set of *background processes* for each instance. They consolidate functions that would otherwise be handled by multiple Oracle programs running for each user process. The background processes asynchronously perform I/O and monitor other Oracle processes to provide increased parallelism for better performance and reliability.
An SGA and the Oracle background processes constitute an Oracle instance.
Each Oracle instance may use several background processes. The names of these processes are DBWR, LGWR, CKPT, SMON, PMON, ARCH, RECO, D*nnn* and LCK*n*. Each background process is described in the following sections.
Database Writer (DBWR) The *Database Writer* writes modified blocks from the database buffer cache to the datafiles. Because of the way Oracle performs logging, DBWR does not need to write blocks when a transaction commits. Instead, DBWR is optimized to minimize disk writes. In general, DBWR writes only when more data

needs to be read into the system global area and too few database buffers are free. The least recently used data is written to the datafiles first.
Log Writer (LGWR) The *Log Writer* writes redo log entries to disk. Redo log data is generated in the redo log buffer of the system global area. As transactions commit and the log buffer fills, LGWR writes redo log entries into an online redo log file.
Checkpoint (CKPT) At specific times, all modified database buffers in the system global area are written to the datafiles by DBWR; this event is called a checkpoint. The *Checkpoint* process is responsible for signalling DBWR at checkpoints and updating all the datafiles and control files of the database to indicate the most recent checkpoint. CKPT is optional; if CKPT is not present, LGWR assumes the responsibilities of CKPT.
System Monitor (SMON) The *system monitor* performs instance recovery at instance startup. In a multiple instance system (one that uses the Parallel Server), SMON of one instance can also perform instance recovery for other instances that have failed. SMON also cleans up temporary segments that are no longer in use and recovers dead transactions skipped during crash and instance recovery because of file-read or offline errors. These transactions are eventually recovered by SMON when the tablespace or file is brought back online. SMON also coalesces free extents within the database to make free space contiguous and easier to allocate.
Process Monitor (PMON) The *process monitor* performs process recovery when a user process fails. PMON is responsible for cleaning up the cache and freeing resources that the process was using. PMON also checks on dispatcher (see below) and server processes and restarts them if they have failed.
Archiver (ARCH) The *archiver* copies the online redo log files to archival storage when they are full. ARCH is active only when a database's redo log is used in ARCHIVELOG mode.
Recoverer (RECO) The *recoverer* is used to resolve distributed transactions that are pending due to a network or system failure in a distributed database. At timed intervals, the local RECO attempts to connect to remote databases and automatically complete the commit or rollback of the local portion of any pending distributed transactions.
Dispatcher (D*nnn*) *Dispatchers* are optional background processes, present only when a multi-threaded server configuration is used. At least one dispatcher process is created for every communication protocol in use (D000, . . ., D*nnn*). Each dispatcher process is responsible for routing requests from connected user processes to available shared server processes and returning the responses back to the appropriate user processes.
Lock (LCK*n*) Up to ten *lock* processes (LCK0, . . ., LCK9) are used for inter-instance locking when the Oracle Parallel Server is used.

The Program Interface

The *program interface* is the mechanism by which a user process communicates with a server process. It serves as a method of standard communication between any client tool or application (such as Oracle Forms) and Oracle software. Its functions are to

- act as a communications mechanism, by formatting data requests, passing data, and trapping and returning errors
- perform conversions and translations of data, particularly between different types of computers or to external user program datatypes

An Example of How Oracle Works

The following example illustrates an Oracle configuration where the user and associated server process are on separate machines (connected via a network).

1. An instance is currently running on the computer that is executing Oracle (often called the *host* or *database server*).
2. A computer used to run an application (a *local machine* or *client workstation*) runs the application in a user process. The client application attempts to establish a connection to the server using the proper SQL*Net driver.
3. The server is running the proper SQL*Net driver. The server detects the connection request from the application and creates a (dedicated) server process on behalf of the user process.
4. The user executes a SQL statement and commits the transaction. For example, the user changes a name in a row of a table.
5. The server process receives the statement and checks the shared pool for any shared SQL area that contains an identical SQL statement. If a shared SQL area is found, the server process checks the user's access privileges to the requested data and the previously existing shared SQL area is used to process the statement; if not, a new shared SQL area is allocated for the statement so that it can be parsed and processed.
6. The server process retrieves any necessary data values from the actual datafile (table) or those stored in the system global area.
7. The server process modifies data in the system global area. The DBWR process writes modified blocks permanently to disk when doing so is efficient. Because the transaction committed, the LGWR process immediately records the transaction in the online redo log file.
8. If the transaction is successful, the server process sends a message across the network to the application. If it is not successful, an appropriate error message is transmitted.

9. Throughout this entire procedure, the other background processes run, watching for conditions that require intervention. In addition, the database server manages other users' transactions and prevents contention between transactions that request the same data.

These steps describe only the most basic level of operations that Oracle performs.

Data Access

This section introduces how Oracle meets the general requirements for a DBMS to do the following:

- adhere to industry accepted standards for a data access language
- control and preserve the consistency of a database's information while manipulating its data
- provide a system for defining and enforcing rules to maintain the integrity of a database's information
- provide high performance

SQL--The Structured Query Language

SQL is a simple, powerful database access language that is the standard language for relational database management systems. The SQL implemented by Oracle Corporation for Oracle is 100 percent compliant with the ANSI/ISO standard SQL data language.

SQL Statements

All operations on the information in an Oracle database are performed using SQL *statements*. A SQL statement is a string of SQL text that is given to Oracle to execute. A statement must be the equivalent of a complete SQL *sentence*, as in

```
SELECT ename, deptno FROM emp;
```

Only a complete SQL statement can be executed, whereas a *sentence fragment*, such as the following, generates an error indicating that more text is required before a SQL statement can execute:

```
SELECT ename
```

A SQL statement can be thought of as a very simple, but powerful, computer program or instruction.

SQL statements are divided into the following categories:

- Data Definition Language (DDL) statements
- Data Manipulation Language (DML) statements
- transaction control statements
- session control statements
- system control statements
- embedded SQL statements

Data Definition Statements (DDL) *DDL statements* define, maintain, and drop objects when they are no longer needed. DDL statements also include statements that permit a user to grant other users the *privileges*, or rights, to access the database and specific objects within the database.

Data Manipulation Statements (DML) *DML statements* manipulate the database's data. For example, querying, inserting, updating, and deleting rows of a table are all DML operations; locking a table or view and examining the execution plan of an SQL statement are also DML operations.

Transaction Control Statements *Transaction control* statements manage the changes made by DML statements. They allow the user or application developer to group changes into logical transactions. Examples include COMMIT, ROLLBACK, and SAVEPOINT.

Session Control Statements *Session control* statements allow a user to control the properties of his current session, including enabling and disabling roles and changing language settings. The two session control statements are ALTER SESSION and SET ROLE.

System Control Statements System control commands change the properties of the Oracle Server instance. The only system control command is ALTER SYSTEM; it allows you to change such settings as the minimum number of shared servers, to kill a session, and to perform other tasks.

Embedded SQL Statements Embedded SQL statements incorporate DDL, DML, and transaction control statements in a procedural language program (such as those used with the Oracle Precompilers). Examples include OPEN, CLOSE, FETCH, and EXECUTE.

Transactions

A *transaction* is a logical unit of work that comprises one or more SQL statements executed by a single user. According to the ANSI/ISO SQL standard, with which Oracle is compatible, a transaction begins with the user's first executable SQL statement. A transaction ends when it is explicitly committed or rolled back (both terms are discussed later in this section) by that user.

Consider a banking database. When a bank customer transfers money from a savings account to a checking account, the transaction might consist of three separate operations: decrease the savings account, increase the checking account, and record the transaction in the transaction journal.
Oracle must guarantee that all three SQL statements are performed to maintain the accounts in proper balance. When something prevents one of the statements in the transaction from executing (such as a hardware failure), the other statements of the transaction must be undone; this is called "rolling back." If an error occurs in making either of the updates, then neither update is made.
Figure 1 - 5 illustrates the banking transaction example.

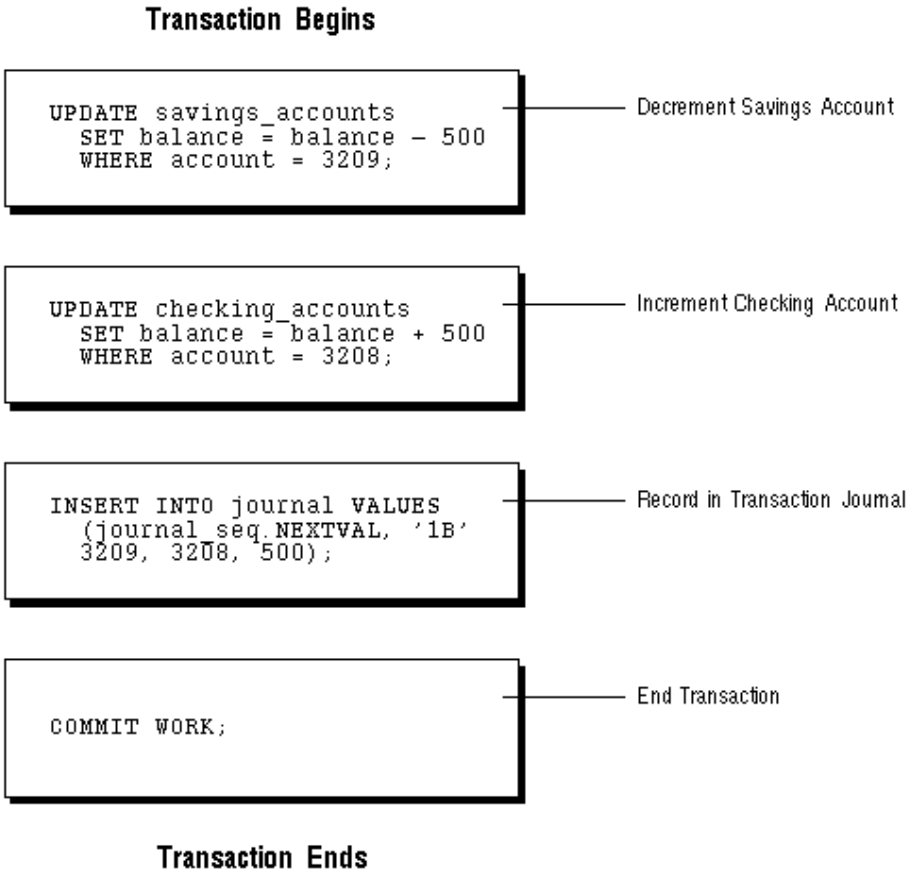

Figure 1 - 5. A Banking Transaction

Committing and Rolling Back Transactions

The changes made by the SQL statements that constitute a transaction can be either committed or rolled back. After a transaction is committed or rolled back, the next transaction begins with the next SQL statement.
Committing a transaction makes permanent the changes resulting from all SQL statements in the transaction. The changes made by the SQL statements of a transaction become visible to other user sessions' transactions that start only after the transaction is committed.
Rolling back a transaction retracts any of the changes resulting from the SQL statements in the transaction. After a transaction is rolled back, the affected data is left unchanged as if the SQL statements in the transaction were never executed.

Savepoints

For long transactions that contain many SQL statements, intermediate markers, or *savepoints*, can be declared. Savepoints can be used to divide a transaction into smaller parts.
By using savepoints, you can arbitrarily mark your work at any point within a long transaction. This allows you the option of later rolling back all work performed from the current point in the transaction to a declared savepoint

within the transaction. For example, you can use savepoints throughout a long complex series of updates, so if you make an error, you do not need to resubmit every statement.

Data Consistency Using Transactions

Transactions provide the database user or application developer with the capability of guaranteeing consistent changes to data, as long as the SQL statements within a transaction are grouped logically. A transaction should consist of all of the necessary parts for one logical unit of work -- no more and no less. Data in all referenced tables are in a consistent state before the transaction begins and after it ends. Transactions should consist of only the SQL statements that comprise one consistent change to the data.
For example, recall the banking example. A transfer of funds between two accounts (the transaction) should include increasing one account (one SQL statement), decreasing another account (one SQL statement), and the record in the transaction journal (one SQL statement). All actions should either fail or succeed together; the credit should not be committed without the debit. Other non-related actions, such as a new deposit to one account, should not be included in the transfer of funds transaction; such statements should be in other transactions.

PL/SQL

PL/SQL is Oracle's procedural language extension to SQL. PL/SQL combines the ease and flexibility of SQL with the procedural functionality of a structured programming language, such as IF ... THEN, WHILE, and LOOP.
When designing a database application, a developer should consider the advantages of using stored PL/SQL:

- Because PL/SQL code can be stored centrally in a database, network traffic between applications and the database is reduced, so application and system performance increases.
- Data access can be controlled by stored PL/SQL code. In this case, users of PL/SQL can access data only as intended by the application developer (unless another access route is granted).
- PL/SQL blocks can be sent by an application to a database, executing complex operations without excessive network traffic.

Even when PL/SQL is not stored in the database, applications can send blocks of PL/SQL to the database rather than individual SQL statements, thereby again reducing network traffic.
The following sections describe the different program units that can be defined and stored centrally in a database.

Procedures and Functions

Procedures and *functions* consist of a set of SQL and PL/SQL statements that are grouped together as a unit to solve a specific problem or perform a set of related tasks. A procedure is created and stored in compiled form in the database and can be executed by a user or a database application. Procedures and functions are identical except that functions always return a single value to the caller, while procedures do not return values to the caller.

Packages

Packages provide a method of encapsulating and storing related procedures, functions, variables, and other package constructs together as a unit in the database. While packages allow the administrator or application developer the ability to organize such routines, they also offer increased functionality (for example, global package variables can be declared and used by any procedure in the package) and performance (for example, all objects of the package are parsed, compiled, and loaded into memory once).

Database Triggers

Oracle allows you to write procedures that are automatically executed as a result of an insert in, update to, or delete from a table. These procedures are called database triggers.
Database triggers can be used in a variety of ways for the information management of your database. For example, they can be used to automate data generation, audit data modifications, enforce complex integrity constraints, and customize complex security authorizations.

Data Integrity

It is very important to guarantee that data adheres to certain business rules, as determined by the database administrator or application developer. For example, assume that a business rule says that no row in the INVENTORY table can contain a numeric value greater than 9 in the SALE_DISCOUNT column. If an INSERT or UPDATE statement attempts to violate this integrity rule, Oracle must roll back the invalid statement and return an error to the application. Oracle provides integrity constraints and database triggers as solutions to manage a database's data integrity rules.

Integrity Constraints

An *integrity constraint* is a declarative way to define a business rule for a column of a table. An integrity constraint is a statement about a table's data that is always true:

- If an integrity constraint is created for a table and some existing table data does not satisfy the constraint, the constraint cannot be enforced.
- After a constraint is defined, if any of the results of a DML statement violate the integrity constraint, the statement is rolled back and an error is returned.

Integrity constraints are defined with a table and are stored as part of the table's definition, centrally in the database's data dictionary, so that all database applications must adhere to the same set of rules. If a rule changes, it need only be changed once at the database level and not many times for each application.
The following integrity constraints are supported by Oracle:
NOT NULL
Disallows nulls (empty entries) in a table's column.
UNIQUE
Disallows duplicate values in a column or set of columns.
PRIMARY KEY
Disallows duplicate values and nulls in a column or set of columns.
FOREIGN KEY
Requires each value in a column or set of columns match a value in a related table's UNIQUE or PRIMARY KEY (FOREIGN KEY integrity constraints also define referential integrity actions that dictate what Oracle should do with dependent data if the data it references is altered).
CHECK
Disallows values that do not satisfy the logical expression of the constraint.
Keys The term "key" is used in the definitions of several types of integrity constraints. A *key* is the column or set of columns included in the definition of certain types of integrity constraints. Keys describe the relationships between the different tables and columns of a relational database. The different types of keys include
primary key
The column or set of columns included in the definition of a table's PRIMARY KEY constraint. A primary key's values uniquely identify the rows in a table. Only one primary key may be defined per table.
unique key
The column or set of columns included in the definition of a UNIQUE constraint.
foreign key
The column or set of columns included in the definition of a referential integrity constraint.
referenced key
The unique key or primary key of the same or different table that is referenced by a foreign key.
Individual values in a key are called *key values*.

Database Triggers

Centralized actions can be defined using a non-declarative approach (writing PL/SQL code) with database triggers. A *database trigger* is a stored procedure that is fired (implicitly executed) when an INSERT, UPDATE, or DELETE statement is issued against the associated table. Database triggers can be used to customize a database management system with such features as value-based auditing and the enforcement of complex security checks and integrity rules. For example, a database trigger might be created to allow a table to be modified only during normal business hours.
Note: While database triggers allow you to define and enforce integrity rules, a database trigger is not the same as an integrity constraint. Among other things, a database trigger defined to enforce an integrity rule does not check data already loaded into a table. Therefore, it is strongly recommended that you use database triggers only when the integrity rule cannot be enforced by integrity constraints.

Data Concurrency and Consistency

This section explains the software mechanisms used by Oracle to fulfill the following important requirements of an information management system:

- Data must be read and modified in a consistent fashion.
- Data concurrency of a multi-user system must be maximized.
- High performance is required for maximum productivity from the many users of the database system.

Concurrency

A primary concern of a multi-user database management system is how to control *concurrency*, or the simultaneous access of the same data by many users. Without adequate concurrency controls, data could be updated or changed improperly, compromising data integrity.

If many people are accessing the same data, one way of managing data concurrency is to make each user wait his or her turn. The goal of a database management system is to reduce that wait so it is either non-existent or negligible to each user. All DML statements should proceed with as little interference as possible and destructive interactions between concurrent transactions must be prevented. Destructive interaction is any interaction that incorrectly updates data or incorrectly alters underlying data structures. Neither performance nor data integrity can be sacrificed.
Oracle resolves such issues by using various types of locks and a multiversion consistency model. Both features are discussed later in this section. These features are based on the concept of a transaction.

Read Consistency

Read consistency, as supported by Oracle, does the following:

- guarantees that the set of data seen by a statement is consistent with respect to a single point-in-time and does not change during statement execution (statement-level read consistency)
- ensures that readers of database data do not wait for writers or other readers of the same data
- ensures that writers of database data do not wait for readers of the same data
- ensures that writers only wait for other writers if they attempt to update identical rows in concurrent transactions

The simplest way to think of Oracle's implementation of read consistency is to imagine each user operating a private copy of the database, hence the multiversion consistency model.

Read Consistency, Rollback Segments, and Transactions

To manage the multiversion consistency model, Oracle must create a read-consistent set of data when a table is being queried (read) and simultaneously updated (written). When an update occurs, the original data values changed by the update are recorded in the database's rollback segments. As long as this update remains part of an uncommitted transaction, any user that later queries the modified data views the original data values -- Oracle uses current information in the system global area and information in the rollback segments to construct a *read-consistent view* of a table's data for a query. Only when a transaction is committed are the changes of the transaction made permanent. Statements, which start *after* the user's transaction is committed, only see the changes made by the committed transaction.
Note that a transaction is key to Oracle's strategy for providing read consistency. This unit of committed (or uncommitted) SQL statements

- dictates the start point for read-consistent views generated on behalf of readers
- controls when modified data can be seen by other transactions of the database for reading or updating

Read-Only Transactions

By default, Oracle guarantees statement-level read consistency. The set of data returned by a single query is consistent with respect to a single point in time. However, in some situations, you may also require transaction-level read consistency -- the ability to run multiple queries within a single transaction, all of which are read-consistent with respect to the same point in time, so that queries in this transaction do not see the effects of intervening committed transactions.
If you want to run a number of queries against multiple tables and if you are doing no updating, you may prefer a *read-only transaction*. After indicating that your transaction is read-only, you can execute as many queries as you like against any table, knowing that the results of each query are consistent with respect to the same point in time.

Locking

Oracle also uses *locks* to control concurrent access to data. Locks are mechanisms intended to prevent destructive interaction between users accessing Oracle data.
Locks are used to achieve two important database goals:
consistency
Ensures that the data a user is viewing or changing is not changed (by other users) until the user is finished with the data.
integrity
Ensures that the database's data and structures reflect all changes made to them in the correct sequence.
Locks guarantee data integrity while allowing maximum concurrent access to the data by unlimited users.

Automatic Locking

Oracle locking is performed automatically and requires no user action. Implicit locking occurs for SQL statements as necessary, depending on the action requested.
Oracle's sophisticated lock manager automatically locks table data at the row level. By locking table data at the row level, contention for the same data is minimized.

Oracle's lock manager maintains several different types of row locks, depending on what type of operation established the lock. In general, there are two types of locks: *exclusive locks* and *share locks*. Only one exclusive lock can be obtained on a resource (such as a row or a table); however, many share locks can be obtained on a single resource. Both exclusive and share locks always allow queries on the locked resource, but prohibit other activity on the resource (such as updates and deletes).

Manual Locking

Under some circumstances, a user may want to override default locking. Oracle allows manual override of automatic locking features at both the row level (by first querying for the rows that will be updated in a subsequent statement) and the table level.

Database Security

Multi-user database systems, such as Oracle, include security features that control how a database is accessed and used. For example, security mechanisms do the following:

- prevent unauthorized database access
- prevent unauthorized access to schema objects
- control disk usage
- control system resource usage (such as CPU time)
- audit user actions

Associated with each database user is a *schema* by the same name. A schema is a logical collection of objects (tables, views, sequences, synonyms, indexes, clusters, procedures, functions, packages, and database links). By default, each database user creates and has access to all objects in the corresponding schema.
Database security can be classified into two distinct categories: system security and data security.
System security includes the mechanisms that control the access and use of the database at the system level. For example, system security includes:

- valid username/password combinations
- the amount of disk space available to the objects of a user
- the resource limits for a user

System security mechanisms check:

- whether a user is authorized to connect to the database
- whether database auditing is active
- which system operations a user can perform

Data security includes the mechanisms that control the access and use of the database at the object level. For example, data security includes

- which users have access to a specific schema object and the specific types of actions allowed for each user on the object (for example, user SCOTT can issue SELECT and INSERT statements but not DELETE statements using the EMP table)
- the actions, if any, that are audited for each schema object

Security Mechanisms

The Oracle Server provides *discretionary access control*, which is a means of restricting access to information based on privileges. The appropriate privilege must be assigned to a user in order for that user to access an object. Appropriately privileged users can grant other users privileges at their discretion; for this reason, this type of security is called "discretionary".
Oracle manages database security using several different facilities:

- database users and schemas
- privileges
- roles
- storage settings and quotas
- resource limits
- auditing

Figure 1 - 6 illustrates the relationships of the different Oracle security facilities, and the following sections provide an overview of users, privileges, and roles.

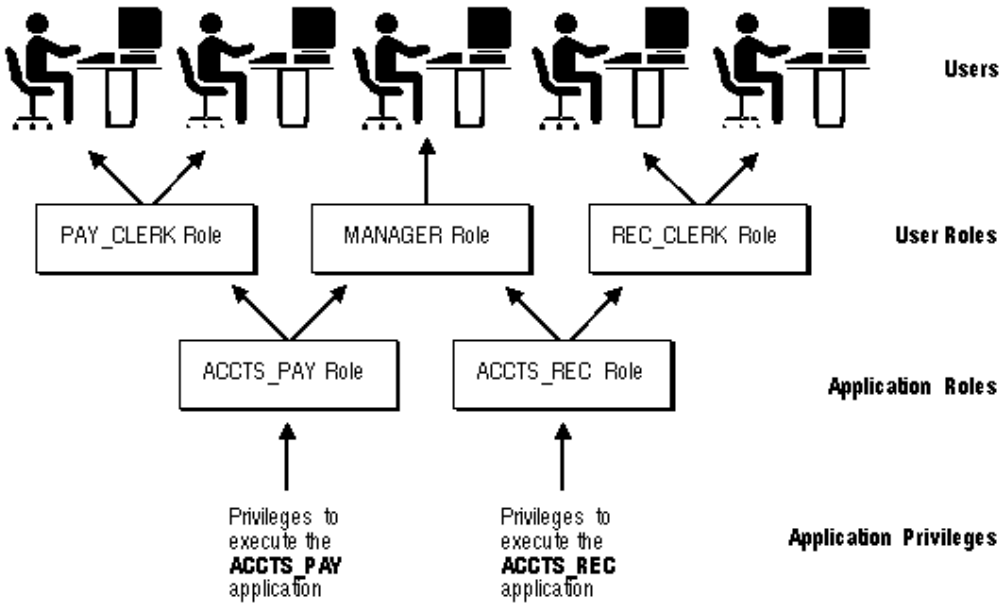

Figure 1 - 6. Oracle Security Features

Database Users and Schemas

Each Oracle database has a list of usernames. To access a database, a user must use a database application and attempt a connection with a valid username of the database. Each username has an associated password to prevent unauthorized use.

Security Domain Each user has a *security domain* -- a set of properties that determine such things as the

- actions (privileges and roles) available to the user
- tablespace quotas (available disk space) for the user
- system resource limits (for example, CPU processing time) for the user

Each property that contributes to a user's security domain is discussed in the following sections.

Privileges

A *privilege* is a right to execute a particular type of SQL statement. Some examples of privileges include the

- right to connect to the database (create a session)
- right to create a table in your schema
- right to select rows from someone else's table
- right to execute someone else's stored procedure

The privileges of an Oracle database can be divided into two distinct categories: system privileges and object privileges.

System Privileges *System privileges* allow users to perform a particular systemwide action or a particular action on a particular type of object. For example, the privileges to create a tablespace or to delete the rows of any table in the database are system privileges. Many system privileges are available only to administrators and application developers because the privileges are very powerful.

Object Privileges *Object privileges* allow users to perform a particular action on a specific object. For example, the privilege to delete rows of a specific table is an object privilege. Object privileges are granted (assigned) to end-users so that they can use a database application to accomplish specific tasks.

Granting Privileges Privileges are granted to users so that users can access and modify data in the database. A user can receive a privilege two different ways:

- Privileges can be granted to users explicitly. For example, the privilege to insert records into the EMP table can be explicitly granted to the user SCOTT.
- Privileges can be granted to *roles* (a named group of privileges), and then the role can be granted to one or more users. For example, the privilege to insert records into the EMP table can be granted to the role named CLERK, which in turn can be granted to the users SCOTT and BRIAN.

Because roles allow for easier and better management of privileges, privileges are normally granted to roles and not to specific users. The following section explains more about roles and their use.

Roles

Oracle provides for easy and controlled privilege management through roles. *Roles* are named groups of related privileges that are granted to users or other roles. The following properties of roles allow for easier privilege management:

- *reduced granting of privileges* -- Rather than explicitly granting the same set of privileges to many users, a database administrator can grant the privileges for a group of related users granted to a role. And then the database administrator can grant the role to each member of the group.
- *dynamic privilege management* -- When the privileges of a group must change, only the privileges of the role need to be modified. The security domains of all users granted the group's role automatically reflect the changes made to the role.
- *selective availability of privileges* -- The roles granted to a user can be selectively enabled (available for use) or disabled (not available for use). This allows specific control of a user's privileges in any given situation.
- *application awareness* -- A database application can be designed to enable and disable selective roles automatically when a user attempts to use the application.

Database administrators often create roles for a database application. The DBA grants an application role all privileges necessary to run the application. The DBA then grants the application role to other roles or users. An application can have several different roles, each granted a different set of privileges that allow for more or less data access while using the application.
The DBA can create a role with a password to prevent unauthorized use of the privileges granted to the role. Typically, an application is designed so that when it starts, it enables the proper role. As a result, an application user does not need to know the password for an application's role.

Storage Settings and Quotas

Oracle provides means for directing and limiting the use of disk space allocated to the database on a per user basis, including default and temporary tablespaces and tablespace quotas.
Default Tablespace Each user is associated with a *default tablespace*. When a user creates a table, index, or cluster and no tablespace is specified to physically contain the object, the user's default tablespace is used if the user has the privilege to create the object and a quota in the specified default tablespace. The default tablespace feature provides Oracle with information to direct space usage in situations where object location is not specified.
Temporary Tablespace Each user has a *temporary tablespace*. When a user executes a SQL statement that requires the creation of temporary segments (such as the creation of an index), the user's temporary tablespace is used. By directing all users' temporary segments to a separate tablespace, the temporary tablespace feature can reduce I/O contention among temporary segments and other types of segments.
Tablespace Quotas Oracle can limit the collective amount of disk space available to the objects in a schema. *Quotas* (space limits) can be set for each tablespace available to a user. The tablespace quota security feature permits selective control over the amount of disk space that can be consumed by the objects of specific schemas.

Profiles and Resource Limits

Each user is assigned a *profile* that specifies limitations on several system resources available to the user, including the

- number of concurrent sessions the user can establish
- CPU processing time
    - available to the user's session
    - available to a single call to Oracle made by a SQL statement
- amount of logical I/O
    - available to the user's session
    - available to a single call to Oracle made by a SQL statement
- amount of idle time for the user's session allowed
- amount of connect time for the user's session allowed

Different profiles can be created and assigned individually to each user of the database. A default profile is present for all users not explicitly assigned a profile. The resource limit feature prevents excessive consumption of global database system resources.

Auditing

Oracle permits selective *auditing* (recorded monitoring) of user actions to aid in the investigation of suspicious database use. Auditing can be performed at three different levels: statement auditing, privilege auditing, and object auditing.
statement auditing

Statement auditing is the auditing of specific SQL statements without regard to specifically named objects. (In addition, database triggers allow a DBA to extend and customize Oracle's built-in auditing features.)
Statement auditing can be broad and audit all users of the system or can be focused to audit only selected users of the system. For example, statement auditing by user can audit connections to and disconnections from the database by the users SCOTT and LORI.
privilege auditing
Privilege auditing is the auditing of the use of powerful system privileges without regard to specifically named objects. Privilege auditing can be broad and audit all users or can be focused to audit only selected users.
object auditing
Object auditing is the auditing of accesses to specific schema objects without regard to user. Object auditing monitors the statements permitted by object privileges, such as SELECT or DELETE statements on a given table.
For all types of auditing, Oracle allows the selective auditing of successful statement executions, unsuccessful statement executions, or both. This allows monitoring of suspicious statements, regardless of whether the user issuing a statement has the appropriate privileges to issue the statement.
The results of audited operations are recorded in a table referred to as the *audit trail*. Predefined views of the audit trail are available so that you can easily retrieve audit records.

Trusted Oracle

Trusted Oracle is Oracle Corporation's multilevel secure database management system product. It is designed to provide the high level of secure data management capabilities required by organizations processing sensitive or classified information. Trusted Oracle is compatible with Oracle base products and applications, and it supports all of the functionality of standard Oracle.
In addition, Trusted Oracle enforces *mandatory access control* (also called *MAC)* across a wide range of multilevel secure operating system environments. Mandatory access control is a means of restricting access to information based on *labels*. A user's label indicates what information a user is permitted to access and the type of access (read or write) that the user is allowed to perform. An object's label indicates the sensitivity of the information that the object contains. A user's label must meet certain criteria, determined by MAC policy, in order for him/her to be allowed to access a labeled object. Because this type of access control is always enforced above any discretionary controls implemented by users, this type of security is called "mandatory".

Database Backup and Recovery

This section covers the structures and software mechanisms used by Oracle to provide

- database recovery required by different types of failures
- flexible recovery operations to suit any situation
- availability of data during backup and recovery operations so that users of the system can continue to work

Why Is Recovery Important?

In every database system, the possibility of a system or hardware failure always exists. Should a failure occur and affect the database, the database must be recovered. The goals after a failure are to ensure that the effects of all committed transactions are reflected in the recovered database and to return to normal operation as quickly as possible while insulating users from problems caused by the failure.

Types of Failures

Several circumstances can halt the operation of an Oracle database. The most common types of failure are described below:
user error
User errors can require a database to be recovered to a point in time before the error occurred. For example, a user might accidentally drop a table. To allow recovery from user errors and accommodate other unique recovery requirements, Oracle provides for exact point-in-time recovery. For example, if a user accidentally drops a table, the database can be recovered to the instant in time before the table was dropped.
statement and process failure
Statement failure occurs when there is a logical failure in the handling of a statement in an Oracle program (for example, the statement is not a valid SQL construction). When statement failure occurs, the effects (if any) of the statement are automatically undone by Oracle and control is returned to the user.
A *process failure* is a failure in a user process accessing Oracle, such as an abnormal disconnection or process termination. The failed user process cannot continue work, although Oracle and other user processes can. The Oracle background process PMON automatically detects the failed user process or is informed of it by SQL*Net.

PMON resolves the problem by rolling back the uncommitted transaction of the user process and releasing any resources that the process was using.
Common problems such as erroneous SQL statement constructions and aborted user processes should never halt the database system as a whole. Furthermore, Oracle automatically performs necessary recovery from uncommitted transaction changes and locked resources with minimal impact on the system or other users.
instance failure
Instance failure occurs when a problem arises that prevents an instance (system global area and background processes) from continuing work. Instance failure may result from a hardware problem such as a power outage, or a software problem such as an operating system crash. When an instance failure occurs, the data in the buffers of the system global area is not written to the datafiles.
Instance failure requires *instance recovery*. Instance recovery is automatically performed by Oracle when the instance is restarted. The redo log is used to recover the committed data in the SGA's database buffers that was lost due to the instance failure.
media (disk) failure
An error can arise when trying to write or read a file that is required to operate the database. This is called *disk failure* because there is a physical problem reading or writing physical files on disk. A common example is a disk head crash, which causes the loss of all files on a disk drive. Different files may be affected by this type of disk failure, including the datafiles, the redo log files, and the control files. Also, because the database instance cannot continue to function properly, the data in the database buffers of the system global area cannot be permanently written to the datafiles.
A disk failure requires *media recovery*. Media recovery restores a database's datafiles so that the information in them corresponds to the most recent time point before the disk failure, including the committed data in memory that was lost because of the failure. To complete a recovery from a disk failure, the following is required: backups of the database's datafiles, and all online and necessary archived redo log files.
Oracle provides for complete and quick recovery from all possible types of hardware failures including disk crashes. Options are provided so that a database can be completely recovered or partially recovered to a specific point in time.
If some datafiles are damaged in a disk failure but most of the database is intact and operational, the database can remain open while the required tablespaces are individually recovered. Therefore, undamaged portions of a database are available for normal use while damaged portions are being recovered.

Structures Used for Recovery

Oracle uses several structures to provide complete recovery from an instance or disk failure: the redo log, rollback segments, a control file, and necessary database backups.

The Redo Log

The *redo log* is a set of files that protect altered database data in memory that has not been written to the datafiles. The redo log can be comprised of two parts: the online redo log and the archived redo log.
The Online Redo Log The *online redo log* is a set of two or more *online redo log files* that record all committed changes made to the database. Whenever a transaction is committed, the corresponding redo entries temporarily stored in redo log buffers of the system global area are written to an online redo log file by the background process LGWR.
The online redo log files are used in a cyclical fashion; for example, if two files constitute the online redo log, the first file is filled, the second file is filled, the first file is reused and filled, the second file is reused and filled, and so on. Each time a file is filled, it is assigned a *log sequence number* to identify the set of redo entries.
To avoid losing the database due to a single point of failure, Oracle can maintain multiple sets of online redo log files. A *multiplexed online redo log* consists of copies of online redo log files physically located on separate disks; changes made to one member of the group are made to all members.
If a disk that contains an online redo log file fails, other copies are still intact and available to Oracle. System operation is not interrupted and the lost online redo log files can be easily recovered using an intact copy.
The Archived Redo Log Optionally, filled online redo files can be archived before being reused, creating an *archived redo log*. *Archived (offline) redo log files* constitute the archived redo log.
The presence or absence of an archived redo log is determined by the mode that the redo log is using:
ARCHIVELOG
The filled online redo log files are archived before they are reused in the cycle.
NOARCHIVELOG
The filled online redo log files are not archived.
In ARCHIVELOG mode, the database can be completely recovered from both instance and disk failure. The database can also be backed up while it is open and available for use. However, additional administrative operations are required to maintain the archived redo log.

If the database's redo log is operated in NOARCHIVELOG mode, the database can be completely recovered from instance failure, but not from a disk failure. Additionally, the database can be backed up only while it is completely closed. Because no archived redo log is created, no extra work is required by the database administrator.

Control Files

The *control files* of a database keep, among other things, information about the file structure of the database and the current log sequence number being written by LGWR. During normal recovery procedures, the information in a control file is used to guide the automated progression of the recovery operation.
Multiplexed Control Files This feature is similar to the multiplexed redo log feature: a number of identical control files may be maintained by Oracle, which updates all of them simultaneously.

Rollback Segments

Rollback segments record rollback information used by several functions of Oracle. During database recovery, after all changes recorded in the redo log have been applied, Oracle uses rollback segment information to undo any uncommitted transactions. Because rollback segments are stored in the database buffers, this important recovery information is automatically protected by the redo log.

Database Backups

Because one or more files can be physically damaged as the result of a disk failure, media recovery requires the restoration of the damaged files from the most recent operating system backup of a database. There are several ways to back up the files of a database.
Full Backups A full backup is an operating system backup of all datafiles, online redo log files, and the control file that constitutes an Oracle database. Full backups are performed when the database is closed and unavailable for use.
Partial Backups A partial backup is an operating system backup of part of a database. The backup of an individual tablespace's datafiles or the backup of a control file are examples of partial backups. Partial backups are useful only when the database's redo log is operated in ARCHIVELOG mode.
A variety of partial backups can be taken to accommodate any backup strategy. For example, you can back up datafiles and control files when the database is open or closed, or when a specific tablespace is online or offline. Because the redo log is operated in ARCHIVELOG mode, additional backups of the redo log are not necessary; the archived redo log is a backup of filled online redo log files.

Basic Recovery Steps

Due to the way in which DBWR writes database buffers to datafiles, at any given point in time, a datafile may contain some data blocks tentatively modified by uncommitted transactions and may not contain some blocks modified by committed transactions. Therefore, two potential situations can result after a failure:

- Blocks containing committed modifications were not written to the datafiles, so the changes may only appear in the redo log. Therefore, the redo log contains committed data that must be applied to the datafiles.
- Since the redo log may have contained data that was not committed, uncommitted transaction changes applied by the redo log during recovery must be erased from the datafiles.

To solve this situation, two separate steps are always used by Oracle during recovery from an instance or media failure: rolling forward and rolling back.

Rolling Forward

The first step of recovery is to *roll forward*, that is, reapply to the datafiles all of the changes recorded in the redo log. Rolling forward proceeds through as many redo log files as necessary to bring the datafiles forward to the required time.
If all needed redo information is online, Oracle performs this recovery step automatically when the database starts. After roll forward, the datafiles contain all committed changes as well as any uncommitted changes that were recorded in the redo log.

Rolling Back

The roll forward is only half of recovery. After the roll forward, any changes that were not committed must be undone. After the redo log files have been applied, then the rollback segments are used to identify and undo transactions that were never committed, yet were recorded in the redo log. This process is called *rolling back*. Oracle completes this step automatically.

Distributed Processing and Distributed Databases

As computer networking becomes more and more prevalent in today's computing environments, database management systems must be able to take advantage of distributed processing and storage capabilities. This section explains the architectural features of Oracle that meet these requirements.

Client/Server Architecture: Distributed Processing

Distributed processing uses more than one processor to divide the processing for a set of related jobs. Distributed processing reduces the processing load on a single processor by allowing different processors to concentrate on a subset of related tasks, thus improving the performance and capabilities of the system as a whole.
An Oracle database system can easily take advantage of distributed processing by using its *client/server architecture*. In this architecture, the database system is divided into two parts: a front-end or a *client* portion and a back-end or a *server* portion.
Client The client portion is the front-end database application and interacts with a user through the keyboard, display, and pointing device such as a mouse. The client portion has no data access responsibilities; it concentrates on requesting, processing, and presenting data managed by the server portion. The client workstation can be optimized for its job. For example, it might not need large disk capacity or it might benefit from graphic capabilities.
Server The server portion runs Oracle software and handles the functions required for concurrent, shared data access. The server portion receives and processes SQL and PL/SQL statements originating from client applications. The computer that manages the server portion can be optimized for its duties. For example, it can have large disk capacity and fast processors.

Distributed Databases

Note: The information in this section regarding distributed updates and two-phase commit applies only for those systems using Oracle with the distributed option.
A *distributed database* is a network of databases managed by multiple database servers that appears to a user as a single logical database. The data of all databases in the distributed database can be simultaneously accessed and modified. The primary benefit of a distributed database is that the data of physically separate databases can be logically combined and potentially made accessible to all users on a network.
Each computer that manages a database in the distributed database is called a *node*. The database to which a user is directly connected is called the *local* database. Any additional databases accessed by this user are called *remote* databases. When a local database accesses a remote database for information, the local database is a client of the remote server (client/server architecture, discussed previously).
While a distributed database allows increased access to a large amount of data across a network, it must also provide the ability to hide the location of the data and the complexity of accessing it across the network. The distributed DBMS must also preserve the advantages of administrating each local database as though it were non-distributed.

Location Transparency

Location transparency occurs when the physical location of data is transparent to the applications and users of a database system. Several Oracle features, such as views, procedures, and synonyms, can provide location transparency. For example, a view that joins table data from several databases provides location transparency because the user of the view does not need to know from where the data originates.

Site Autonomy

Site autonomy means that each database participating in a distributed database is administered separately and independently from the other databases, as though each database were a non-networked database. Although each database can work with others, they are distinct, separate systems that are cared for individually.

Distributed Data Manipulation

The Oracle distributed database architecture supports all DML operations, including queries, inserts, updates, and deletes of remote table data. To access remote data, a reference is made including the remote object's global object name -- no coding or complex syntax is required to access remote data. For example, to query a table named EMP in the remote database named SALES, you reference the table's global object name:

```
SELECT * FROM emp@sales;
```

Two-Phase Commit

Oracle provides the same assurance of data consistency in a distributed environment as in a non-distributed environment. Oracle provides this assurance using the transaction model and a *two-phase commit mechanism*. As in non-distributed systems, transactions should be carefully planned to include a logical set of SQL statements that should all succeed or fail as a unit. Oracle's two-phase commit mechanism guarantees that no matter what type of

system or network failure might occur, a distributed transaction either commits on all involved nodes or rolls back on all involved nodes to maintain data consistency across the global distributed database.
Complete Transparency to Database Users The Oracle two-phase commit mechanism is completely transparent to users that issue distributed transactions. A simple COMMIT statement denoting the end of a transaction automatically triggers the two-phase commit mechanism to commit the transaction; no coding or complex statement syntax is required to include distributed transactions within the body of a database application.
Automatic Recovery from System or Network Failures The RECO (recoverer) background process automatically resolves the outcome of *in-doubt distributed transactions* -- distributed transactions in which the commit was interrupted by any type of system or network failure. After the failure is repaired and communication is reestablished, the RECO of each local Oracle Server automatically commits or rolls back any in-doubt distributed transactions consistently on all involved nodes.
Optional Manual Override Capability In the event of a long-term failure, Oracle allows each local administrator to manually commit or roll back any distributed transactions that are in doubt as a result of the failure. This option allows the local database administrator to free up any locked resources that may be held indefinitely as a result of the long-term failure.
Facilities for Distributed Recovery If a database must be recovered to a point in the past, Oracle's recovery facilities allow database administrators at other sites to return their databases to the earlier point in time also. This ensures that the global database remains consistent.

Table Replication

Note: The information in this section applies only to Oracle with the distributed or advanced replication options.
Distributed database systems often locally replicate remote tables that are frequently queried by local users. By having copies of heavily accessed data on several nodes, the distributed database does not need to send information across a network repeatedly, thus helping to maximize the performance of the database application.
Data can be replicated using snapshots or replicated master tables. Replicated master tables require the replication option.

Oracle and SQL*Net

*SQL*Net* is Oracle's mechanism for interfacing with the communication protocols used by the networks that facilitate distributed processing and distributed databases. Communication protocols define the way that data is transmitted and received on a network. In a networked environment, an Oracle database server communicates with client workstations and other Oracle database servers using Oracle software called SQL*Net. SQL*Net supports communications on all major network protocols, ranging from those supported by PC LANs to those used by the largest of mainframe computer systems.
Using SQL*Net, the application developer does not have to be concerned with supporting network communications in a database application. If a new protocol is used, the database administrator makes some minor changes, while the application requires no modifications and continues to function.