CHAPTER 1

Getting Started with the Oracle Architecture
Oracle Database 10g is an evolutionary, if not revolutionary, step from the previous release of Oracle 9i. Not only is Oracle 10g more feature rich, it is easier to manage with more tools to help the Oracle DBA “set it and forget it.” Part I of this book covers the basics of the Oracle architecture and lays the foundation for deploying a successful Oracle infrastructure by giving practical advice for a new installation or upgrading from a previous release of Oracle. To provide a good foundation for the Oracle 10g software, server hardware and operating system configuration issues are covered in the relevant sections.

In Part II of this book, we will cover several areas relevant to the day-to-day maintenance and operation of an Oracle 10g database. The first chapter in Part II discusses the requirements that a DBA needs to gather long before the install CD is mounted on the server. Successive chapters deal with ways the DBA can manage disk space, CPU usage, and adjust Oracle parameters to optimize the server’s resources, using a variety of tools at the DBA’s disposal for monitoring database performance. Transaction management is greatly simplified by Automated Undo Management (AUM), an Oracle Database feature introduced in Oracle 9i and enhanced in Oracle 10g.

Part III of this book focuses on the high availability aspects of Oracle 10g. This includes using Oracle’s Recovery Manager (RMAN) to perform and automate database backups and recovery, along with other features, such as Oracle Data Guard, to provide a reliable and easy way to recover from a database failure. Last, but certainly not least, we will show how Oracle 10g Real Application Clusters (RAC) can at the same time provide extreme scalability and transparent failover capabilities to a database environment.

In Part IV of this book, we will cover a variety of issues revolving around Networked Oracle. Not only will we cover how Oracle Net can be configured in an N-tier environment, but also how we manage large and distributed databases that may reside in neighboring cities or around the world.

In this chapter, we cover the basics of Oracle Database 10g, highlighting many of the features we will cover in the rest of the book as well as the basics of installing Oracle 10g using Oracle Universal Installer (OUI) and the Database Configuration Assistant (DBCA). We will take a tour of the elements that comprise an instance of Oracle 10g, ranging from memory structures to disk structures, initialization parameters, tables, indexes, and PL/SQL. Each of these elements plays a large role in making Oracle 10g a highly scalable, available, and secure environment.

An Overview of Databases and Instances
Although the terms “database” and “instance” are often used interchangeably, they are quite different. They are very distinct entities in an Oracle datacenter, as you shall see in the following sections.

Databases
A database is a collection of data on disk in one or more files on a database server that collects and maintains related information. The database consists of various physical and logical structures, the table being the most important logical structure in the database. A table consists of rows and columns containing related data. At a minimum, a database must have at least tables to store useful information. Figure 1-1 shows a sample table containing four rows and three columns. The data
in each row of the table is related: Each row contains information about a particular employee in the company.

In addition, a database provides a level of security to prevent unauthorized access to the data. Oracle Database 10g provides many mechanisms to facilitate the security necessary to keep confidential data confidential. Oracle Security and access control are covered in more detail in Chapter 10.

Files comprising a database fall into two broad categories: database files and non-database files. The distinction lies in what kind of data is stored in each. Database files contain data and metadata; non-database files contain initialization parameters, logging information, and so forth. Database files are critical to the ongoing operation of the database on a moment-by-moment basis. Each of these physical storage structures is discussed later in the section titled “Oracle Physical Storage Structures.”

## Instances

The main components of a typical enterprise server are one or more CPUs, disk space, and memory. Whereas the Oracle database is stored on a server’s disk, an Oracle instance exists in the server’s memory. An Oracle instance is composed of a large block of memory allocated in an area called the System Global Area (SGA), along with a number of background processes that interact between the SGA and the database files on disk.

In an Oracle Real Application Cluster (RAC), more than one instance will use the same database. Although the instances that share the database can be on the same server, most likely the instances will be on separate servers that are connected by a high-speed interconnect and access a database that resides on a specialized RAID-enabled disk subsystem. More details on how a RAC installation is configured are provided in Chapter 11.
**Oracle Logical Storage Structures**

The datafiles in an Oracle database are grouped together into one or more tablespaces. Within each tablespace, the logical database structures, such as tables and indexes, are segments that are further subdivided into extents and blocks. This logical subdivision of storage allows Oracle to have more efficient control over disk space usage. Figure 1-2 shows the relationship between the logical storage structures in a database.

**Tablespaces**

An Oracle *tablespace* consists of one or more datafiles; a datafile can be a part of one and only one tablespace. For an installation of Oracle 10g, a minimum of two tablespaces are created: the SYSTEM tablespace and the SYSAUX tablespace.

Oracle 10g allows you to create a special kind of tablespace called a *bigfile tablespace*, which can be as large as 8EB (exabytes, or a million terabytes). Using bigfiles makes tablespace management completely transparent to the DBA; in other words, the DBA can manage the tablespace as a unit without worrying about the size and structure of the underlying datafiles.

Using Oracle Managed Files (OMF) can make tablespace datafile management even easier. With OMF, the DBA specifies one or more locations in the file system where datafiles, control files, and redo log files will reside, and Oracle automatically handles the naming and management of these files. We discuss OMF in more detail in Chapter 4.

If a tablespace is *temporary*, the tablespace itself is permanent; only the segments saved in the tablespace are temporary. A temporary tablespace can be used for sorting operations and as a
working area for building indexes. Dedicating a tablespace for these kinds of operations helps to reduce the I/O contention between temporary segments and permanent segments stored in another tablespace, such as tables.

Tablespaces can be either dictionary managed or locally managed. In a dictionary-managed tablespace, extent management is recorded in data dictionary tables. Therefore, even if all application tables are in the USERS tablespace, the SYSTEM tablespace will still be accessed for managing DML on application tables. Because all users and applications must use the SYSTEM tablespace for extent management, this creates a potential bottleneck for write-intensive applications. In a locally managed tablespace, Oracle maintains a bitmap in each datafile of the tablespace to track space availability. Only quotas are managed in the data dictionary, dramatically reducing the contention for data dictionary tables.

As of Oracle9i, if the SYSTEM tablespace is locally managed, then all other tablespaces must be locally managed if both read and write operations are to be performed on them. Dictionary-managed tablespaces must be read-only in databases with a locally-managed SYSTEM tablespace.

### Blocks

A database block is the smallest unit of storage in the Oracle database. The size of a block is a specific number of bytes of storage within a given tablespace within the database.

A block is usually a multiple of the operating system block size to facilitate efficient disk I/O. The default block size is specified by the Oracle initialization parameter DB_BLOCK_SIZE. As many as four other block sizes may be defined for other tablespaces in the database, although the blocks in the SYSTEM, SYSAUX, and any temporary tablespaces must be of the size DB_BLOCK_SIZE.

### Extents

The extent is the next level of logical grouping in the database. An extent consists of one or more database blocks. When a database object is enlarged, the space added to the object is allocated as an extent.

### Segments

The next level of logical grouping in a database is the segment. A segment is a group of extents that comprise a database object that is treated as a unit, such as a table or index. As a result, this is typically the smallest unit of storage that an end user of the database will deal with. Four types of segments are found in an Oracle database: data segments, index segments, temporary segments, and rollback segments.

#### Data Segment

Every table in the database resides in a single data segment, consisting of one or more extents; more than one segment is allocated for a table if it is a partitioned table or a clustered table. Partitioned and clustered tables are discussed later in this chapter.

#### Index Segment

Each index is stored in its own index segment. As with partitioned tables, each partition of a partitioned index is stored in its own segment.
Temporary Segment
When a user’s SQL statement needs disk space to complete an operation, such as a sorting operation that cannot fit in memory, a temporary segment is allocated. Temporary segments exist only for the duration of the SQL statement.

Rollback Segment
As of Oracle 10g, rollback segments only exist in the SYSTEM tablespace, and typically the DBA does not need to maintain the SYSTEM rollback segment. In previous Oracle releases, a rollback segment was created to save the previous values of a database DML operation in case the transaction was rolled back, and to maintain the “before” image data to provide read-consistent views of table data for other users accessing the table. Rollback segments were also used during database recovery for rolling back uncommitted transactions that were active when the database instance crashed or terminated unexpectedly.

In Oracle 10g, Automatic Undo Management handles the automatic allocation and management of rollback segments within an undo tablespace. Within an undo tablespace, the undo segments are structured similarly to rollback segments, except that the details of how these segments are managed is under control of Oracle, instead of being managed (often inefficiently) by the DBA. Automatic undo segments were available starting with Oracle9i, but manually managed rollback segments are still available in Oracle 10g. However, this functionality is deprecated as of Oracle 10g, and will no longer be available in future releases. Automatic Undo Management is discussed in detail in Chapter 7.

Oracle Logical Database Structures
In this section, we will be covering the highlights of all major logical database structures, starting with tables and indexes. Next, we discuss the variety of datatypes we can use to define the columns of a table. When we create a table with columns, we can place restrictions, or constraints, on the columns of the table.

One of the many reasons we use a Relational Database Management System (RDBMS) to manage our data is to leverage the security and auditing features of the Oracle database. We will review the ways we can segregate access to the database by user or by the object being accessed.

We’ll also touch upon many other logical structures that can be defined by either the DBA or the user, including synonyms, links to external files, and links to other databases.

Tables
A table is the basic unit of storage in an Oracle database. Without any tables, a database has no value to an enterprise. Regardless of the type of table, data in a table is stored in rows and columns, similar to how data is stored in a spreadsheet. But that is where the similarity ends. The robustness of a database table due to the surrounding reliability, integrity, and scalability of the Oracle database makes a spreadsheet a poor second choice when deciding on a place to store critical information.

In this section, we will review the many different types of tables in the Oracle database and how they can satisfy most every data-storage need for an organization. More details on how to choose between these types of tables for a particular application, and how to manage them, can be found in Chapter 5 and Chapter 8.
Relational Tables

A relational table is the most common type of table in a database. A relational table is heap-organized; in other words, the rows in the table are stored in no particular order. In the `create table` command, you can specify the clause `organization heap` to define a heap-organized table, but because this is the default, the clause can be omitted.

Each row of a table contains one or more columns; each column has a datatype and a length. As of Oracle version 8, a column may also contain a user-defined object type, a nested table, or a VARRAY. In addition, a table can be defined as an object table. We will review object tables and objects later in this section.

The built-in Oracle datatypes are presented in Table 1-1.

<table>
<thead>
<tr>
<th>Oracle Built-in Datatype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VARCHAR2 (size) [BYTE</td>
<td>CHAR]</td>
</tr>
<tr>
<td>NVARCHAR2(size)</td>
<td>A variable-length character string with a maximum length of 4000 bytes.</td>
</tr>
<tr>
<td>NUMBER(p,s)</td>
<td>A number with a precision (p) and scale (s). The precision ranges from 1 to 38, and the scale can be from –84 to 127.</td>
</tr>
<tr>
<td>LONG</td>
<td>A variable-length character data with a length up to 2GB (2^{31} – 1).</td>
</tr>
<tr>
<td>DATE</td>
<td>Date values from January 1st, 4712 B.C. to December 31st, 9999 A.D.</td>
</tr>
<tr>
<td>BINARY_FLOAT</td>
<td>A 32-bit floating point number.</td>
</tr>
<tr>
<td>BINARY_DOUBLE</td>
<td>A 64-bit floating point number.</td>
</tr>
<tr>
<td>TIMESTAMP (fractional_seconds)</td>
<td>Year, month, day, hour, minute, second, and fractional seconds. Value of fractional_seconds can range from 0 to 9; in other words, up to one billionth of a second precision. The default is 6 (one millionth).</td>
</tr>
<tr>
<td>TIMESTAMP (fractional_seconds) WITH TIME ZONE</td>
<td>Contains a TIMESTAMP value in addition to a time zone displacement value. Time zone displacement can be an offset from UTC (such as ‘-06:00’) or a region name (e.g., ‘US/Central’).</td>
</tr>
<tr>
<td>TIMESTAMP (fractional_seconds) WITH LOCAL TIME ZONE</td>
<td>Similar to TIMESTAMP WITH TIMEZONE, except that (1) data is normalized to the database time zone when it is stored and (2) when retrieving columns with this datatype, the user sees the data in the session’s time zone.</td>
</tr>
<tr>
<td>INTERVAL YEAR (year_precision) TO MONTH</td>
<td>Stores a time period in years and months. The value of year_precision is the number of digits in the YEAR field.</td>
</tr>
<tr>
<td>INTERVAL DAY (day_precision) TO SECOND (fractional_seconds_precision)</td>
<td>Stores a period of time as days, hours, minutes, seconds, and fractional seconds. The value for day_precision is from 0 to 9, with a default of 2. The value of fractional_seconds_precision is similar to the fractional seconds in a TIMESTAMP value; the range is from 0 to 9, with a default of 6.</td>
</tr>
<tr>
<td>RAW(size)</td>
<td>Raw binary data, with a maximum size of 2000 bytes.</td>
</tr>
</tbody>
</table>

**TABLE 1-1. Oracle Built-in Datatypes**
Oracle also supports ANSI-compatible datatypes; the mapping between the ANSI datatypes and Oracle datatypes is provided in Table 1-2.

### Temporary Tables
Temporary tables have been available since Oracle8i. They are temporary in the sense of the data that is stored in the table, not in the definition of the table itself. The command `create global temporary table` creates a temporary table.

As long as other users have permissions to the table itself, they may perform select statements or Data Manipulation Language Commands (DML), such as insert, update, or delete, on a temporary table. However, each user sees their own and only their own data in the table. When a user truncates a temporary table, only the data that they inserted is removed from the table.

There are two different flavors of temporary data in a temporary table: temporary for the duration of the transaction, and temporary for the duration of the session. The longevity of the temporary data is controlled by the on commit clause; on commit delete rows removes all rows from the temporary table when a commit or rollback is issued, and on commit preserve rows keeps the rows in the table beyond the transaction boundary. However, when the user’s session is terminated, all of the user’s rows in the temporary table are removed.
There are a few other things to keep in mind when using temporary tables. Although you can create an index on a temporary table, the entries in the index are dropped along with the data rows, as with a regular table. Also, due to the temporary nature of the data in a temporary table, no redo information is generated for DML on temporary tables; however, undo information is created in the undo tablespace.

**Index Organized Tables**

As you will find out later in the subsection on indexes, creating an index makes finding a particular row in a table more efficient. However, this adds a bit of overhead, because the database must maintain the data rows and the index entries for the table. What if your table does not have many columns, and access to the table occurs primarily on a single column? In this case, an index organized table (IOT) might be the right solution. An IOT stores rows of a table in a B-tree index, where each node of the B-tree index contains the keyed (indexed) column along with one or more non-indexed columns.

The most obvious advantage of an IOT is that only one storage structure needs to be maintained instead of two; similarly, the values for the primary key of the table are only stored once in an IOT, versus twice in a regular table.

There are, however, a few disadvantages to using an IOT. Some tables, such as tables for logging events, may not need a primary key, or any keys for that matter; an IOT must have a primary key. Also, IOTs cannot be a member of a cluster. Finally, an IOT might not be the best solution for a table if there are a large number of columns in the table, and many of the columns are frequently accessed when table rows are retrieved.
Object Tables

Since Oracle8, the Oracle Database has supported many object-oriented features in the database. User-defined types, along with any defined methods for these object types, can make an implementation of an object-oriented (OO) development project in Oracle seamless.

Object tables have rows that are themselves objects, or instantiations of type definitions. Rows in an object table can be referenced by object ID (OID), in contrast to a primary key in a relational, or regular, table; however, object tables can still have both primary and unique keys, just as relational tables do.

Let’s say, for example, that you are creating a Human Resources (HR) system from scratch, so you have the flexibility to design the database from an entirely OO point of view. The first step is to define an employee object, or type, by creating the type:

```sql
create type PERS_TYP as object
(Last_Name varchar2(45),
First_Name varchar2(30),
Middle_Initial char(1),
Surname varchar2(10),
SSN varchar2(15));
```

In this particular case, you’re not creating any methods with the PERS_TYP object, but by default Oracle creates a constructor method for the type that has the same name as the type itself (in this case, PERS_TYP). To create an object table as a collection of PERS_TYP objects, you can use the familiar `create table` syntax, as follows:

```sql
create table pers of pers_typ;
```

To add an instance of an object to the object table, you specify the constructor method in the `insert` command:

```sql
insert into pers
values(pers_typ('Judd','Dawn','R','Dr.','123-45-6789'));
```

References to instances of the PERS_TYP object can be stored in other tables as REF objects, and data from the PERS table can be retrieved without a direct reference to the PERS table itself.

More examples of how objects can be used to implement an object-oriented design project can be found in Chapter 5.

External Tables

External tables were introduced in Oracle9i. In a nutshell, external tables allow a user to access a data source, such as a text file, as if it was a table in the database. The metadata for the table is stored within the Oracle data dictionary, but the contents of the table are stored externally.

The definition for an external table contains two parts. The first and most familiar part is the definition of the table from the database user’s point of view. This definition looks like any typical definition that you’d see in a `create table` statement.

The second part, however, is what differentiates an external table from a regular table. This is where the mapping between the database columns and the external data source occurs—what column(s) the data element starts in, how wide the column is, and whether the format of the external column is character or binary. The syntax for the default type of external table,
ORACLE_LOADER, is virtually identical to that of a control file in SQL*Loader. This is one of the advantages of external tables; the user only needs to know how to access a standard database table to get to the external file.

There are a few drawbacks, however, to using external tables. Indexes cannot be created on an external table, and no inserts, updates, or deletes can be performed on external tables. These drawbacks are minor when considering the advantages of using external tables for loading native database tables, for example, in a data warehouse environment.

Clustered Tables
If two or more tables are frequently accessed together (for example, an order table and a line item detail table), then creating a clustered table might be a good way to boost the performance of queries that reference those tables. In the case of an order table with an associated line-item detail table, the order header information could be stored in the same block as the line-item detail records, thus reducing the amount of I/O needed to retrieve the order and line-item information.

Clustered tables also reduce the amount of space needed to store the columns the two tables have in common, also known as a cluster key value. The cluster key value is also stored in a cluster index. The cluster index operates much like a traditional index in that it will improve queries against the clustered tables when accessed by the cluster key value. In our example with orders and line items, the order number is only stored once, instead of repeating for each line-item detail row.

The advantages to clustering a table are reduced if frequent insert, update, and delete operations occur on the table relative to the number of select statements against the table. In addition, frequent queries against individual tables in the cluster may also reduce the benefits of clustering the tables in the first place.

Hash Clusters
A special type of clustered table, a hash cluster, operates much like a regular clustered table, except that instead of using a cluster index, a hash cluster uses a hashing function to store and retrieve rows in a table. The total estimated amount of space needed for the table is allocated when the table is created, given the number of hash keys specified during the creation of the cluster. In our order-entry example, let’s assume that our Oracle database needs to mirror the legacy data-entry system, which reuses order numbers on a periodic basis. Also, the order number is always a six-digit number. We might create the cluster for orders as in the following example:

```sql
create cluster order_cluster (order_number number(6)) size 50
    hash is order_number hashkeys 1000000;
create table cust_order (
    order_number number(6) primary key,
    order_date date,
    customer_number number)
cluster order_cluster(order_number);
```

Hash clusters have performance benefits when rows are selected from a table using an equality comparison, as in this example:

```sql
select order_number, order_date from cust_order
    where order_number = 196811;
```
Typically, this kind of query will retrieve the row with only one I/O if the number of hashkeys is high enough and the hash is clause, containing the hashing function, produces an evenly distributed hash key.

Sorted Hash Clusters
Sorted hash clusters are new in Oracle 10g. They are similar to regular hash clusters in that a hashing function is used to locate a row in a table. However, in addition, sorted hash clusters allow rows in the table to be stored by one or more columns of the table in ascending order. This allows the data to be processed more quickly for applications that lend themselves to First In First Out (FIFO) processing.

You create sorted hash clusters using the same syntax as regular clustered tables, with the addition of the SORT positional parameter after the column definitions within the cluster. Here is an example of creating a table in a sorted hash cluster:

```sql
create table order_detail (
    order_number number,
    order_timestamp timestamp sort,
    customer_number number)
cluster order_detail_cluster (
    order_number,
    order_timestamp);
```

Due to the FIFO nature of a sorted hash cluster, when orders are accessed by order_number the oldest orders are retrieved first based on the value of order_timestamp.

Partitioned Tables
Partitioning a table (or index, as you will see in the next section) helps make a large table more manageable. A table may be partitioned, or even subpartitioned, into smaller pieces. From an application point of view, partitioning is transparent (that is, no explicit references to a particular partition are necessary in any end-user SQL). The only effect that a user may notice is that queries against the partitioned table using criteria in the where clause that matches the partitioning scheme run a lot faster!

There are many advantages to partitioning from a DBA point of view. If one partition of a table is on a corrupted disk volume, the other partitions in the table are still available for user queries while the damaged volume is being repaired. Similarly, backups of partitions can occur over a period of days, one partition at a time, rather than requiring a single backup of the entire table.

Partitions are one of three types: range partitioned, hash partitioned, or, as of Oracle9i, list partitioned. Each row in a partitioned table can exist in one, and only one, partition. The partition key directs the row to the proper partition; the partition key can be a composite key of up to 16 columns in the table. There are a few minor restrictions on the types of tables that can be partitioned; for example, a table containing a LONG or LONG RAW column cannot be partitioned.

**TIP**
Oracle Corporation recommends that any table greater than 2GB in size be seriously considered for partitioning.
No matter what type of partitioning scheme is in use, each member of a partitioned table must have the same logical attributes, such as column names, datatypes, constraints, and so forth. The physical attributes for each partition, however, can be different depending on its size and location on disk. The key is that the partitioned table must be logically consistent from an application or user point of view.

**Range Partitions**  A range partition is a partition whose partition key falls within a certain range. For example, visits to the corporate e-commerce website can be assigned to a partition based on the date of the visit, with one partition per quarter. A website visit on May 25th, 2004 will be recorded in the partition with the name FY2004Q2, whereas a website visit on December 2nd, 2004 will be recorded in the partition with the name FY2004Q4.

**List Partitions**  A list partition is a partition whose partition key falls within groups of distinct values. For example, sales by region of the country may create a partition for NY, CT, MA, and VT, and another partition for IL, WI, IA, and MN. Any sales from elsewhere in the world may be assigned to its own partition when the state code is missing.

**Hash Partitions**  A hash partition assigns a row to a partition based on a hashing function, specifying the column or columns used in the hashing function, but not explicitly assigning the partition, only specifying how many partitions are available. Oracle will assign the row to a partition and ensure a balanced distribution of rows in each partition.

Hash partitions are useful when there is no clear list or range-partitioning scheme given the types of columns in the table, or when the relative sizes of the partitions change frequently, requiring repeated manual adjustments to the partitioning scheme.

**Composite Partitions**  Even further refinement of the partitioning process is available with composite partitions. For example, a table may be partitioned by range, and within each range, subpartitioned by list or by hash.

**Partitioned Indexes**  The indexes on a table may also be partitioned, either matching the partition scheme of the table being indexed (local indexes) or partitioned independently from the partition scheme of the table (global indexes). Local partitioned indexes have the advantage of increased availability of the index when partition operations occur; for example, archiving and dropping the partition FY2002Q4 and its local index will not affect index availability for the other partitions in the table.

**Constraints**  An Oracle constraint is a rule or rules that can be defined on one or more columns in a table to help enforce a business rule. For example, a constraint can enforce the business rule that an employee’s starting salary must be at least $25,000.00. Another example of a constraint enforcing a business rule is to require that if a new employee is assigned a department (although they need not be assigned to a particular department right away), the department number must be valid and exist in the DEPT table.

Six types of data integrity rules can be applied to table columns: null rule, unique column values, primary key values, referential integrity values, complex in-line integrity, and trigger-based integrity. We will touch upon each of these briefly in the following sections.
All the constraints on a table are defined either when the table is created or when the table is altered at the column level, except for triggers, which are defined based on which DML operation is being performed on the table. Constraints may be enabled or disabled at creation or at any point of time in the future; when a constraint is either enabled or disabled (using the keyword enable or disable), existing data in the table may or may not have to be validated (using the keyword validate or novalidate) against the constraint, depending on the business rules in effect.

For example, a table in an automaker’s database named CAR_INFO containing new automobile data needs a new constraint on the AIRBAG_QTY column, where the value of this column must not be NULL and must have a value that is at least 1 for all new vehicles. However, this table contains data for model years before air bags were required, and as a result, this column is either 0 or NULL. One solution, in this case, would be to create a constraint on the AIRBAG_QTY table to enforce the new rule for new rows added to the table, but not to validate the constraint for existing rows.

Here is a table created with all constraint types. Each constraint is reviewed in the following subsections.

```sql
create table CUST_ORDER
(Order_Number NUMBER(6) PRIMARY KEY,
Order_Date DATE NOT NULL,
Delivery_Date DATE,
Warehouse_Number NUMBER DEFAULT 12,
Customer_Number NUMBER NOT NULL,
Order_Line_Item_Qty NUMBER CHECK (Order_Line_Item_Qty < 100),
UPS_Tracking_Number VARCHAR2(50) UNIQUE,
foreign key (Customer_Number) references CUSTOMER(Customer_Number));
```

**Null Rule**

The NOT NULL constraint prevents NULL values from being entered into the Order_Date or Customer_Number column. This makes a lot of sense from a business rule point of view: Every order must have an order date, and an order doesn’t make any sense unless a customer places it.

Note that a NULL value in a column doesn’t mean that the value is blank or zero; rather, the value does not exist. A NULL value is not equal to anything, not even another NULL value. This concept is important when using SQL queries against columns that may have NULL values.

**Unique Column Values**

The UNIQUE integrity constraint ensures that a column or group of columns (in a composite constraint) is unique throughout the table. In the preceding example, the UPS_Tracking_Number column will not contain duplicate values.

To enforce the constraint, Oracle will create a unique index on the UPS_Tracking_Number column. If there is already a valid unique index on the column, Oracle will use that index to enforce the constraint.

A column with a UNIQUE constraint may also be declared as NOT NULL. If the column is not declared with the NOT NULL constraint, then any number of rows may contain NULL values, as long as the remaining rows have unique values in this column.

In a composite unique constraint that allows NULLs in one or more columns, the columns that are not NULL determine whether the constraint is being satisfied. The NULL column always satisfies the constraint, because a NULL value is not equal to anything.
Primary Key Values
The PRIMARY KEY integrity constraint is the most common type of constraint found in a database table. At most, only one primary key constraint can exist on a table. The column or columns that comprise the primary key cannot have NULL values.

In the preceding example, the Order_Number column is the primary key. A unique index is created to enforce the constraint; if a useable unique index already exists for the column, the primary key constraint uses that index.

Referential Integrity Values
The referential integrity or FOREIGN KEY constraint is more complicated than the others we have covered so far because it relies on another table to restrict what values can be entered into the column with the referential integrity constraint.

In the preceding example, a FOREIGN KEY is declared on the Customer_Number column; any values entered into this column must also exist in the Customer_Number column of another table (in this case, the CUSTOMER table).

As with other constraints that allow NULL values, a column with a referential integrity constraint can be NULL without requiring that the referenced column contain a NULL value.

Furthermore, a FOREIGN KEY constraint can be self-referential. In an EMPLOYEE table whose primary key is Employee_Number, the Manager_Number column can have a FOREIGN KEY declared against the Employee_Number column in the same table. This allows for the creation of a reporting hierarchy within the EMPLOYEE table itself.

Indexes should almost always be declared on a FOREIGN KEY column to improve performance; the only exception to this rule is when the referenced primary or unique key in the parent table is never updated or deleted.

Complex In-Line Integrity
More complex business rules may be enforced at the column level by using a CHECK constraint. In the preceding example, the Order_Line_Item_Qty column must never exceed 99.

A CHECK constraint can use other columns in the row being inserted or updated to evaluate the constraint. For example, a constraint on the STATE_CD column would allow NULL values only if the COUNTRY_CD column is not USA. In addition, the constraint can use literal values and built-in functions such as TO_CHAR or TO_DATE, as long as these functions operate on literals or columns in the table.

Multiple CHECK constraints are allowed on a column. All the CHECK constraints must evaluate to TRUE to allow a value to be entered in the column. For example, we could modify the preceding CHECK constraint to ensure that Order_Line_Item_Qty is greater than 0 in addition to being less than 100.

Trigger-Based Integrity
If the business rules are too complex to implement using unique constraints, a database trigger can be created on a table using the create trigger command along with a block of PL/SQL code to enforce the business rule.

Triggers are required to enforce referential integrity constraints when the referenced table exists in a different database. Triggers are also useful for many things outside the realm of constraint checking (auditing access to a table, for example).

Database triggers are covered in-depth in Chapter 18.
Indexes
An Oracle index allows faster access to rows in a table when a small subset of the rows will be retrieved from the table. An index stores the value of the column or columns being indexed, along with the physical RowID of the row containing the indexed value, except for index organized tables (IOTs), which use the primary key as a logical RowID. Once a match is found in the index, the RowID in the index points to the exact location of the table row: which file, which block within the file, and which row within the block.

Indexes are created on a single column or multiple columns. Index entries are stored in a B-tree structure so that traversing the index to find the key value of the row uses very few I/O operations. An index may serve a dual purpose in the case of a unique index: Not only will it speed the search for the row, but it enforces a unique or primary key constraint on the indexed column. Entries within an index are automatically updated whenever the contents of a table row are inserted, updated, or deleted. When a table is dropped, all indexes created on the table are also automatically dropped.

Several types of indexes are available in Oracle, each suitable for a particular type of table, access method, or application environment. We will present the highlights and features of the most common index types in the following subsections.

Unique Indexes
A unique index is the most common form of B*Tree index. It is often used to enforce the primary key constraint of a table. Unique indexes ensure that duplicate values will not exist in the column or columns being indexed. A unique index may be created on a column in the EMPLOYEE table for the Social Security Number because there should not be any duplicates in this column. However, some employees may not have a Social Security Number, so this column would contain a NULL value.

Non-Unique Indexes
A non-unique index helps speed access to a table without enforcing uniqueness. For example, we can create a non-unique index on the Last_Name column of the EMPLOYEE table to speed up our searches by last name, but we would certainly have many duplicates for any given last name.

A non-unique B*Tree index is created on a column by default if no other keywords are specified in a CREATE INDEX statement.

Reverse Key Indexes
A reverse key index is a special kind of index used typically in an OLTP (online transaction processing) environment. In a reverse key index, all the bytes in each column’s key value of the index are reversed. The reverse keyword specifies a reverse key index in the create index command. Here is an example of creating a reverse key index:

```sql
create index IB_LINE_ITEM_ORDER_NUMBER
on LINE_ITEM(Order_Number) REVERSE;
```

If an order number of 123459 is placed, the reverse key index stores the order number as 954321. Inserts into the table are distributed across all leaf keys in the index, reducing the contention among several writers all doing inserts of new rows. A reverse key index also reduces the potential for these “hot spots” in an OLTP environment if orders are queried or modified soon after they are placed.
Function-Based Indexes

A function-based index is similar to a standard B*Tree index, except that a transformation of a column or columns, declared as an expression, is stored in the index instead of the columns themselves.

Function-based indexes are useful in cases where names and addresses might be stored in the database as mixed case. A regular index on a column containing the value ‘SmiTh’ would not return any values if the search criterion was ‘Smith’. On the other hand, if the index stored the last names in all uppercase, all searches on last names could use uppercase. Here is an example of creating a function-based index on the Last_Name column of the EMPLOYEE table:

```sql
create index up_name on employee(upper(Last_Name));
```

As a result, searches using queries such as the following will use the index we just created instead of doing a full table scan:

```sql
select Employee_Number, Last_Name, First_Name from employee
where upper(Last_Name) = 'SMITH';
```

Bitmap Indexes

A bitmap index has a significantly different structure from a B*Tree index in the leaf node of the index. It stores one string of bits for each possible value (the cardinality) of the column being indexed. The length of the string of bits is the same as the number of rows in the table being indexed.

In addition to saving a tremendous amount of space compared to traditional indexes, a bitmap index can provide dramatic improvements in response time because Oracle can quickly remove potential rows from a query containing multiple `where` clauses long before the table itself needs to be accessed. Multiple bitmaps can use logical `and` and `or` operations to determine which rows to access from the table.

Although a bitmap index can be used on any column in a table, it is most efficient when the column being indexed has a low cardinality, or number of distinct values. For example, the Gender column in the PERS table will either be NULL, M, or F. The bitmap index on the Gender column will have only three bitmaps stored in the index. On the other hand, a bitmap index on the Last_Name column will have close to the same number of bitmap strings as rows in the table itself! The queries looking for a particular last name will most likely take less time if a full table scan is performed instead of using an index. In this case, a traditional B*Tree non-unique index makes more sense.

A variation of bitmap indexes called bitmap join indexes creates a bitmap index on a table column that is frequently joined with one or more other tables on the same column. This provides tremendous benefits in a data warehouse environment where a bitmap join index is created on a fact table and one or more dimension tables, essentially prejoining those tables and saving CPU and I/O resources when an actual join is performed.

**NOTE**

Bitmap indexes are only available in the Enterprise Edition of Oracle 10g.
20 Oracle Database 10g DBA Handbook

Views
Views allow users to see a customized presentation of the data in a single table or even a join between many tables. A view is also known as a stored query—the query details underlying the view are hidden from the user of the view. A regular view does not store any data, only the definition, and the underlying query is run every time the view is accessed. Extensions to a regular view, called a materialized view, allows the results of the query to be stored along with the definition of the query to speed processing, among other benefits. Object views, like traditional views, hide the details of the underlying table joins and allow object-oriented development and processing to occur in the database while the underlying tables are still in a relational format.

In the following subsections, we’ll review the basics of the types of views a typical user will create and use on a regular basis.

Regular Views
A regular view, or more commonly referred to as a view, is not allocated any storage; only its definition, a query, is stored in the data dictionary. The tables in the query underlying the view are called base tables; each base table in a view can be further defined as a view.

The advantages of a view are many. Views hide data complexity—a senior analyst can define a view containing the EMPLOYEE, DEPARTMENT, and SALARY tables to make it easier for upper management to retrieve information about employee salaries by using a select statement against what appears to be a table but is actually a view containing a query that joins the EMPLOYEE, DEPARTMENT, and SALARY tables.

Views can also be used to enforce security. A view on the EMPLOYEE table called EMP_INFO may contain all columns except for salary, and the view can be defined as read only to prevent updates to the table:

```
create view EMP_INFO as
  select Employee_Number, Last_Name,
  First_Name, Middle_Initial, Surname
from EMPLOYEE
with READ ONLY;
```

Without the read only clause, it is possible to update or add rows to a view, even to a view containing multiple tables. There are some constructs in a view that prevent it from being updatable, such as having a distinct operator, an aggregate function, or a group by clause.

When Oracle processes a query containing a view, it substitutes the underlying query definition in the user’s select statement and processes the resulting query as if the view did not exist. As a result, the benefits of any existing indexes on the base tables are not lost when a view is used.

Materialized Views
In some ways, a materialized view is very similar to a regular view: The definition of the view is stored in the data dictionary, and the view hides the details of the underlying base query from the user. But that is where the similarities end. A materialized view also allocates space in a database segment to hold the result set from the execution of the base query.

A materialized view can be used to replicate a read-only copy of table to another database, with the same column definitions and data as the base table. This is the simplest implementation of a materialized view. To enhance the response time when a materialized view needs to be refreshed, a materialized view log can be created to refresh the materialized view. Otherwise, a full refresh...
is required when a refresh is required—the results of the base query must be run in their entirety to refresh the materialized view. The materialized view log facilitates incremental updates of the materialized views.

In a data warehouse environment, materialized views can store aggregated data from a group by rollup or a group by cube query. If the appropriate initialization parameter values are set, such as QUERY_REWRITE_ENABLED, and the query itself allows for query rewrites (with the query rewrite clause), then any query that appears to do the same kind of aggregation as the materialized view will automatically use the materialized view instead of running the original query.

Regardless of the type of materialized view, it can be refreshed automatically when a committed transaction occurs in the base table, or it can be refreshed on demand.

Materialized views have many similarities to indexes, in that they are directly tied to a table and take up space, they must be refreshed when the base tables are updated, their existence is virtually transparent to the user, and they can aid in optimizing queries by using an alternate access path to return the results of a query.

More details on how to use materialized views in a distributed environment can be found in Chapter 18.

Object Views
Object-oriented (OO) application development environments are becoming increasingly prevalent, and the Oracle 10g database fully supports the implementation of objects and methods natively in the database. However, a migration from a purely relational database environment to a purely OO database environment is not an easy transition to make; few organizations have the time and resources to build a new system from the ground up. Oracle 10g makes the transition easier with object views. Object views allow the object-oriented applications to see the data as a collection of objects that have attributes and methods, while the legacy systems can still run batch jobs against the INVENTORY table. Object views can simulate abstract datatypes, object identifiers (OIDs), and references that a purely OO database environment would provide.

As with regular views, you can use instead of triggers in the view definition to allow DML against the view by running a block of PL/SQL code instead of the actual DML statement supplied by the user or application.

Users and Schemas
Access to the database is granted to a database account known as a user. A user may exist in the database without owning any objects. However, if the user creates and owns objects in the database, those objects are part of a schema that has the same name as the database user. A schema can own any type of object in the database: tables, indexes, sequences, views, and so forth. The schema owner or DBA can grant access to these objects to other database users. The user always has full privileges and control over the objects in the user’s schema.

When a user is created by the DBA (or by any other user with the create user system privilege), a number of other characteristics can be assigned to the user, such as which tablespaces are available to the user for creating objects, and whether the password is preexpired.

Users can be authenticated in the database with three methods: database authentication, operating system authentication, and network authentication. With database authentication, the encrypted password for the user is stored in the database. In contrast, operating system authentication makes an assumption that a user who is already authenticated by an operating system connection has the same privileges as a user with the same or similar name (depending on the value of the
Profiles
Database resources are not unlimited; therefore, a DBA must manage and allocate resources among all database users. Some examples of database resources are CPU time, concurrent sessions, logical reads, and connect time.

A database profile is a named set of resource limits that can be assigned to a user. After Oracle is installed, the DEFAULT profile exists and is assigned to any user not explicitly assigned a profile. The DBA can add new profiles or change the DEFAULT profile to suit the needs of the enterprise. The initial values for the DEFAULT profile allow for unlimited use of all database resources.

Sequences
An Oracle sequence assigns sequential numbers, guaranteed to be unique unless the sequence is re-created or reset. It produces a series of unique numbers in a multiuser environment without the overhead of disk locking or any special I/O calls, other than what’s involved in loading the sequence into the shared pool.

Sequences can generate numbers up to 38 digits in length; the series of numbers can be ascending or descending, the interval can be any user-specified value, and Oracle can cache blocks of numbers from a sequence in memory for even faster performance.

The numbers from sequences are guaranteed to be unique, but not necessarily sequential. If a block of numbers is cached, and the instance is restarted, or a transaction that uses a number from a sequence is rolled back, the next call to retrieve a number from the sequence will not return the number that was not used in the original reference to the sequence.

Synonyms
An Oracle synonym is simply an alias to a database object, to simplify references to database objects and to hide the details of the source of the database objects. Synonyms can be assigned to tables, views, materialized views, sequences, procedures, functions, and packages. Like views, a synonym allocates no space in the database, other than its definition in the data dictionary.

Synonyms can be either public or private. A private synonym is defined in the schema of a user and is available only to the user. A public synonym is usually created by a DBA and is automatically available for use by any database user.

**TIP**
After creating a public synonym, make sure the users of the synonym have the correct privileges to the object referenced by the synonym.

When referencing a database object, Oracle first checks whether the object exists in the user’s schema. If no such object exists, Oracle checks for a private synonym. If there is no private synonym, Oracle checks for a public synonym. If there is no public synonym, an error is returned.
PL/SQL
Oracle PL/SQL is Oracle’s procedural language extension to SQL. PL/SQL is useful when the standard DML and select statements cannot produce the desired results in an easy fashion because of the lack of the procedural elements found in a traditional third-generation language such as C++ and Ada. As of Oracle9i, the SQL processing engine is shared between SQL and PL/SQL, which means that all new features added to SQL are automatically available to PL/SQL.

In the next few sections, we’ll take a whirlwind tour of the benefits of using Oracle PL/SQL.

Procedures/Functions
PL/SQL procedures and functions are examples of PL/SQL named blocks. A PL/SQL block is a sequence of PL/SQL statements treated as a unit for the purposes of execution, and it contains up to three sections: a variable declaration section, an executable section, and an exception section.

The difference between a procedure and function is that a function will return a single value to a calling program such as a SQL select statement. A procedure, on the other hand, does not return a value, only a status code. However, procedures may have one or many variables that can be set and returned as part of the argument list to the procedure.

Procedures and functions have many advantages in a database environment. Procedures are compiled and stored in the data dictionary once; when more than one user needs to call the procedure, it is already compiled, and only one copy of the stored procedure exists in the shared pool. In addition, network traffic is reduced, even if the procedural features of PL/SQL are not used. One PL/SQL call uses up much less network bandwidth than several SQL select and insert statements sent separately over the network, not to mention the reparsing that occurs for each statement sent over the network.

Packages
PL/SQL packages group together related functions and procedures, along with common variables and cursors. Packages consist of two parts: a package specification and a package body. In the package specification, the methods and attributes of the package are exposed; the implementation of the methods along with any private methods and attributes are hidden in the package body.

Using a package instead of a standalone procedure or function allows the embedded procedure or function to be changed without invalidating any objects that refer to elements of the package specification, thus avoiding recompilation of the objects that reference the package.

Triggers
Triggers are a specialized type of a PL/SQL or Java block of code that is executed, or triggered, when a specified event occurs. The types of events can be DML statements on a table or view, DDL statements, and even database events such as startup or shutdown. The specified trigger can be refined to execute on a particular event for a particular user as part of an auditing strategy.

Triggers are extremely useful in a distributed environment to simulate a foreign key relationship between tables that do not exist in the same database. They are also very useful in implementing complex integrity rules that cannot be defined using the built-in Oracle constraint types.

More information on how triggers can be used in a robust distributed environment can be found in Chapter 18.
External File Access

In addition to external tables, there are a number of other ways Oracle can access external files:

- From SQL*Plus, either by accessing an external script containing other SQL commands to be run or by sending the output from a SQL*Plus spool command to a file in the operating system’s file system.
- Text information can be read or written from a PL/SQL procedure using the UTL_FILE built-in package; similarly, dbms_output calls within a PL/SQL procedure can generate text messages and diagnostics that can be captured by another application and saved to a text file.
- External data can be referenced by the BFILE datatype. A BFILE datatype is a pointer to an external binary file. Before BFILES can be used in a database, a directory alias needs to be created with the create directory command that specifies a prefix containing the full directory path where the BFILE target is stored.
- DBMS_PIPE can communicate with any 3GL language that Oracle supports, such as C++, Ada, Java, or COBOL, and exchange information.
- UTL_MAIL, a new package in Oracle 10g, allows a PL/SQL application to send e-mails without knowing how to use the underlying SMTP protocol stack.

When using an external file as a data source, for either input or output, a number of cautions are in order. The following should be carefully considered before you use an external data source:

- The database data and the external data may be frequently out of synch when one of the data sources changes without synchronizing with the other.
- It is important to make sure that the backups of the two data sources occur at nearly the same time to ensure that the recovery of one data source will keep the two data sources in synch.
- Script files may contain passwords; many organizations forbid the plain-text representation of any user account in a script file. In this situation, operating system validation may be a good alternative for user authentication.
- You should review the security of files located in a directory that is referenced by each DIRECTORY object. Extreme security measures on database objects are mitigated by lax security on referenced operating system files.

Database Links and Remote Databases

Database links allow an Oracle database to reference objects stored outside of the local database. The command create database link creates the path to a remote database, which in turn allows access to objects in the remote database. A database link wraps together the name of the remote database, a method for connecting to the remote database, and a username/password combination to authenticate the connection to the remote database. In some ways, a database link is similar to a database synonym: A database link can be public or private, and it provides a convenient shorthand way to access another set of resources. The main difference is that the resource is outside of the
database instead of in the same database, and therefore requires more information to resolve the reference. The other difference is that a synonym is a reference to a specific object, whereas a database link is a defined path used to access any number of objects in a remote database.

For links to work between databases in a distributed environment, the global database name of each database in the domain must be different. Therefore, it is important to assign the initialization parameters DB_NAME and DB_DOMAIN correctly.

To make using database links even easier, a synonym can be assigned to a database link, to make the table access even more transparent; the user does not know if the synonym accesses an object locally or on a distributed database. The object can move to a different remote database, or to the local database, and the synonym name can remain the same, making access to the object transparent to users.

How database links to remote databases are leveraged in a distributed environment is covered further in Chapter 18.

Oracle Physical Storage Structures
The Oracle database uses a number of physical storage structures on disk to hold and manage the data from user transactions. Some of these storage structures, such as the datafiles, redo log files, and archived redo log files, hold actual user data; other structures, such as control files, maintain the state of the database objects, and text-based alert and trace files contain logging information for both routine events and error conditions in the database. Figure 1-3 shows the relationship between these physical structures and the logical storage structures we reviewed in the previous section “Oracle Logical Database Structures.”

Datafiles
Every Oracle database must contain at least one datafile. One Oracle datafile corresponds to one physical operating system file on disk. Each datafile in an Oracle database is a member of one and only one tablespace; a tablespace, however, can consist of many datafiles.

An Oracle datafile may automatically expand when it runs out of space, if the DBA created the datafile with the AUTOEXTEND parameter. The DBA can also limit the amount of expansion for a given datafile by using the MAXSIZE parameter. In any case, the size of the datafile is ultimately limited by the disk volume on which it resides.

TIP
The DBA often has to decide whether to allocate one datafile that can autoextend indefinitely or to allocate many smaller datafiles with a limit to how much each can extend. Although the performance of each solution is likely very similar, it is probably a better idea to stick with more datafiles that are each less than 2GB in size. It is a lot easier to move around relatively smaller files, and some file systems may limit the size of an individual file to 2GB anyway. Also, if you need to temporarily move all the datafiles for a tablespece to another server, it is often easier to find several volumes, each with enough space to hold one of the datafiles, rather than one volume with enough space to hold a single datafile that is 25GB.
The datafile is the ultimate resting place for all data in the database. Frequently accessed blocks in a datafile are cached in memory; similarly, new data blocks are not immediately written out to the datafile but rather are written to the datafile depending on when the database writer process is active. Before a user’s transaction is considered complete, however, the transaction’s changes are written to the redo log files.

**Redo Log Files**

Whenever data is added, removed, or changed in a table, index, or other Oracle object, an entry is written to the current *redo log file*. Every Oracle database must have at least two redo log files, because Oracle reuses redo log files in a circular fashion. When one redo log file is filled with *redo log entries*, the current log file is marked as ACTIVE, if it is still needed for instance recovery, or INACTIVE, if it is not needed for instance recovery; the next log file in the sequence is reused from the beginning of the file and is marked as CURRENT.
Ideally, the information in a redo log file is never used. However, when a power failure occurs, or some other server failure causes the Oracle instance to fail, the new or updated data blocks in the database buffer cache may not yet have been written to the datafiles. When the Oracle instance is restarted, the entries in the redo log file are applied to the database datafiles in a roll forward operation, to restore the state of the database up to the point where the failure occurred.

To be able to recover from the loss of one redo log file within a redo log group, multiple copies of a redo log file can exist on different physical disks. Later in this chapter, you will see how redo log files, archived log files, and control files can be multiplexed to ensure the availability and data integrity of the Oracle database.

Control Files
Every Oracle database has at least one control file that maintains the metadata of the database (in other words, data about the physical structure of the database itself). Among other things, it contains the name of the database, when the database was created, and the names and locations of all datafiles and redo log files. In addition, the control file maintains information used by Recovery Manager (RMAN), such as the persistent RMAN settings and the types of backups that have been performed on the database. RMAN is covered in depth in Chapter 13. Whenever any changes are made to the structure of the database, the information about the changes is immediately reflected in the control file.

Because the control file is so critical to the operation of the database, it can also be multiplexed. However, no matter how many copies of the control file are associated with an instance, only one of the control files is designated as primary for purposes of retrieving database metadata.

The alter database backup controlfile to trace command is another way to back up the control file. It produces a SQL script that can be used to re-create the database control file in case all multiplexed binary versions of the control file are lost due to a catastrophic failure.

This trace file can also be used, for example, to re-create a control file if the database needs to be renamed, or to change various database limits that could not otherwise be changed without re-creating the entire database.

Archived Log Files
An Oracle database can operate in one of two modes: archivelog or noarchivelog mode. When the database is in noarchivelog mode, the circular reuse of the redo log files (also known as the online redo log files) means that redo entries (the contents of previous transactions) are no longer available in case of a failure to a disk drive or another media-related failure. Operating in noarchivelog mode does protect the integrity of the database in the event of an instance failure or system crash, because all transactions that are committed but not yet written to the datafiles are available in the online redo log files.

In contrast, archivelog mode sends a filled redo log file to one or more specified destinations and can be available to reconstruct the database at any given point in time in the event that a database media failure occurs. For example, if the disk drive containing the datafiles crashes, the contents of the database can be recovered to a point in time before the crash, given a recent backup of the datafiles and the redo log files that were generated since the backup occurred.

The use of multiple archived log destinations for filled redo log files is critical for one of Oracle’s high-availability features known as Oracle Data Guard, formerly known as Oracle Standby Database. Oracle Data Guard is covered in detail in Chapter 14.
Initialization Parameter Files

When a database instance starts, the memory for the Oracle instance is allocated, and one of two types of initialization parameter files is opened: either a text-based file called init<SID>.ora (known generically as init.ora or a PFILE) or a server parameter file (otherwise known as an SPFILE). The instance first looks for an SPFILE in the default location for the operating system ($ORACLE_HOME/dbs on Unix, for example) as either spfile<SID>.ora or spfile.ora. If neither of these files exists, the instance looks for a PFILE with the name init<SID>.ora. Alternatively, the startup command can explicitly specify a PFILE to use for startup.

Initialization parameter files, regardless of the format, specify file locations for trace files, control files, filled redo log files, and so forth. They also set limits on the sizes of the various structures in the System Global Area (SGA) as well as how many users can connect to the database simultaneously.

Until Oracle9i, using the init.ora file was the only way to specify initialization parameters for the instance. Although it is easy to edit with a text editor, it has some drawbacks. If a dynamic system parameter is changed at the command line with the alter system command, the DBA must remember to change the init.ora file so that the new parameter value will be in effect the next time the instance is restarted.

An SPFILE makes parameter management easier and more effective for the DBA. If an SPFILE is in use for the running instance, any alter system command that changes an initialization parameter can change the initialization parameter automatically in the SPFILE, change it only for the running instance, or both. No editing of the SPFILE is necessary, or even possible without corrupting the SPFILE itself.

Although you cannot mirror a parameter file or SPFILE per se, you can back up an SPFILE to an init.ora file, and both the init.ora and the SPFILE for the Oracle instance should be backed up using conventional operating system commands or using Recovery Manager in the case of an SPFILE.

When the DBCA is used to create a database, an SPFILE is created by default.

Alert and Trace Log Files

When things go wrong, Oracle can and often does write messages to the alert log and, in the case of background processes or user sessions, trace log files.

The alert log file, located in the directory specified by the initialization parameter BACKGROUND_DUMP_DEST, contains both routine status messages as well as error conditions. When the database is started up or shut down, a message is recorded in the alert log, along with a list of initialization parameters that are different from their default values. In addition, any alter database or alter system commands issued by the DBA are recorded. Operations involving tablespaces and their datafiles are recorded here, too, such as adding a tablespace, dropping a tablespace, and adding a datafile to a tablespace. Error conditions, such as tablespaces running out of space, corrupted redo logs, and so forth, are also recorded here.

The trace files for the Oracle instance background processes are also located in BACKGROUND_DUMP_DEST. For example, the trace files for PMON and SMON contain an entry when an error occurs or when SMON needs to perform instance recovery; the trace files for QMON contain informational messages when it spawns a new process.

Trace files are also created for individual user sessions or connections to the database. These trace files are located in the directory specified by the initialization parameter USER_DUMP_DEST. Trace files for user processes are created in two situations: The first is when some type of error occurs in a user session because of a privilege problem, running out of space, and so forth. In the
second situation, a trace file can be created explicitly with the command `alter session set sql_trace=true`. Trace information is generated for each SQL statement that the user executes, which can be helpful when tuning a user’s SQL statement.

The alert log file can be deleted or renamed at any time; it is re-created the next time an alert log message is generated. The DBA will often set up a daily batch job (either through an operating system mechanism or using Oracle Enterprise Manager’s scheduler) to rename and archive the alert log on a daily basis.

**Backup Files**
Backup files can originate from a number of sources, such as operating system copy commands or Oracle Recovery Manager (RMAN). If the DBA performs a “cold” backup (see the section titled “Backup/Recovery Overview” for more details on backup types), the backup files are simply operating system copies of the datafiles, redo log files, control files, archived redo log files, and so forth.

In addition to bit-for-bit image copies of datafiles (the default in RMAN), RMAN can generate full and incremental backups of datafiles, control files, redo log files, archived log files, and SPFILEs that are in a special format, called backupsets, only readable by RMAN. RMAN backupset backups are generally smaller than the original datafiles because RMAN does not back up unused blocks.

**Oracle Managed Files**
Oracle Managed Files (OMF), introduced in Oracle version 9i, makes the DBA’s job easier by automating the creation and removal of the datafiles that comprise the logical structures in the database.

Without OMF, a DBA might drop a tablespace and forget to remove the underlying operating system files. This makes inefficient use of disk resources, and it unnecessarily increases backup time for datafiles that are no longer needed by the database.

OMF is well suited for small databases with a low number of users and a part-time DBA, where optimal configuration of a production database is not necessary.

**Password Files**
An Oracle password file is a file within the Oracle administrative or software directory structure on disk used to authenticate Oracle system administrators for tasks such as creating a database or starting up and shutting down the database. The privileges granted through this file are the SYSDBA and SYSOPER privileges. Authenticating any other type of user is done within the database itself; because the database may be shut down or not mounted, another form of administrator authentication is necessary in these cases.

The Oracle command-line utility `orapwd` creates a password file if one does not exist or is damaged. Because of the extremely high privileges granted via this file, it should be stored in a secure directory location that is not available to anyone except for DBAs and operating system administrators. Once this file is created, the initialization parameter `REMOTE_LOGIN_PASSWORDFILE` should be set to `EXCLUSIVE` to allow users other than SYS to use the password file.

**TIP**
Create at least one user other than SYS or SYSTEM who has DBA privileges for daily administrative tasks. If there is more than one DBA administering a database, each DBA should have their own account with DBA privileges.
Alternatively, authentication for the SYSDBA and SYSOPER privileges can be done with OS authentication; in this case, a password file does not have to be created, and the initialization parameter REMOTE_LOGIN_PASSWORDFILE is set to NONE.

Multiplexing Database Files

To minimize the possibility of losing a control file or a redo log file, multiplexing of database files reduces or eliminates data-loss problems caused by media failures. Multiplexing can be somewhat automated by using an Automatic Storage Management (ASM) instance, available starting in Oracle 10g. For a more budget-conscious enterprise, control files and redo log files can be multiplexed manually.

Automatic Storage Management

Using Automatic Storage Management is a multiplexing solution that automates the layout of datafiles, control files, and redo log files by distributing them across all available disks. When new disks are added to the ASM cluster, the database files are automatically redistributed across all disk volumes for optimal performance. The multiplexing features of an ASM cluster minimize the possibility of data loss and are generally more effective than a manual scheme that places critical files and backups on different physical drives.

Manual Multiplexing

Without a RAID or ASM solution, you can still provide some safeguards for your critical database files by setting some initialization parameters and providing an additional location for control files, redo log files, and archived redo log files.

Control Files

Control files can be multiplexed immediately when the database is created, or they can be multiplexed at any time later with a few extra steps to manually copy them to multiple destinations. Up to eight copies of a control file can be multiplexed.

Whether you multiplex the control files when the database is created or you multiplex them later, the initialization parameter value for CONTROL_FILES is the same. In the following example, three copies of the control file will be multiplexed:

```
Alter system
set control_files = '/u01/oracle/whse2/ctrlwhse1.ctl',
'/u02/oracle/whse2/ctrlwhse2.ctl',
'/u03/oracle/whse2/ctrlwhse3.ctl',
scope=spfile;
```

If the database is created with this parameter set, the three control files will be created, and there are no additional steps.

If you want to add another multiplexed location, you need to edit the initialization parameter file and add another location to the CONTROL_FILES parameter. If an SPFILE is being used instead of an init.ora file, then use a command similar to the following to change the CONTROL_FILES parameter:
alter system
  set control_files = '/u01/oracle/whse2/ctrlwhse1.ctl,
  /u02/oracle/whse2/ctrlwhse2.ctl,
  /u03/oracle/whse2/ctrlwhse3.ctl'
scope=spfile;

The other possible values for SCOPE in the alter system command are MEMORY and BOTH. Specifying either one of these for SCOPE returns an error, because the CONTROL_FILES parameter cannot be changed for the running instance, only for the next restart of the instance. Therefore, only the SPFILE is changed.

In either case, the next step is to shut down the database. Copy the control file to the new destinations, as specified in CONTROL_FILES, and restart the database. You can always verify the names and locations of the control files by looking in one of the data dictionary views:

select value from v$spparameter where name='control_files';

This query will return one row for each multiplexed copy of the control file. In addition, the view V$CONTROLFILE contains one row for each copy of the control file along with a status.

Redo Log Files

Redo log files are multiplexed by changing a set of redo log files into a redo log file group. In a default Oracle installation, a set of three redo log files is created. As you learned in the previous section on redo log files, after each log file is filled, it starts filling the next in sequence. After the third is filled, the first one is reused. To change the set of three redo log files to a group, we can add one or more identical files as a companion to each of the existing redo log files. After the groups are created, the redo log entries are concurrently written to the group of redo log files. When the group of redo log files is filled, it begins to write redo entries to the next group. Figure 1-4 shows how a set of four redo log files can be multiplexed with four groups, each group containing three members.

Adding a member to a redo log group is very straightforward. In the alter database command, we specify the name of the new file and the group to add it to. The new file is created with the same size as the other members in the group:

alter database
  add logfile member '/u05/oracle/dc2/log_3d.dbf'
  to group 3;

If the redo log files are filling up faster than they can be archived, one possible solution is to add another redo log group. Here is an example of how to add a fifth redo log group to the set of redo log groups in Figure 1-4:

alter database
  add logfile group 5
  ('/u02/oracle/dc2/log_3a.dbf',
   '/u03/oracle/dc2/log_3b.dbf',
   '/u04/oracle/dc2/log_3c.dbf') size 10m;

All members of a redo log group must be the same size. However, the log file sizes between groups may be different. In addition, redo log groups may have a different number of members. In the preceding example, we started with four redo log groups, added an extra member to redo log group 3 (for a total of four members), and added a fifth redo log group with three members.
As of Oracle 10g, you can use the Redo Logfile Sizing Advisor to assist in determining the optimal size for redo log files to avoid excessive I/O activity or bottlenecks. See Chapter 8 for more information on how to use the Redo Logfile Sizing Advisor.

**Archived Redo Log Files**

If the database is in archivelog mode, redo log files are copied to a specified location before they can be reused in the redo log switch cycle.

**Oracle Memory Structures**

Oracle uses the server’s physical memory to hold many things for an Oracle instance: the Oracle executable code itself, session information, individual processes associated with the database, and information shared between processes (such as locks on database objects). In addition, the memory structures contain user and data dictionary SQL statements, along with cached information that is eventually permanently stored on disk, such as data blocks from database segments and information about completed transactions in the database. The data area allocated for an Oracle instance is called the System Global Area (SGA). The Oracle executables reside in the software code area. In addition, an area called the Program Global Area (PGA) is private to each server and background process; one PGA is allocated for each process. Figure 1-5 shows the relationships between these Oracle memory structures.
System Global Area

The System Global Area is a group of shared memory structures for an Oracle instance, shared by the users of the database instance. When an Oracle instance is started, memory is allocated for the SGA based on the values specified in the initialization parameter file or hard-coded in the Oracle software. Many of the parameters that control the size of the various parts of the SGA are dynamic; however, if the parameter SGA_MAX_SIZE is specified, the total size of all SGA areas must not exceed the value of SGA_MAX_SIZE. If SGA_MAX_SIZE is not specified, but the parameter SGA_TARGET is specified, Oracle automatically adjusts the sizes of the SGA components so that the total amount of memory allocated is equal to SGA_TARGET. SGA_TARGET is a dynamic parameter; it can be changed while the instance is running.

Memory in the SGA is allocated in units of granules. A granule can be either 4MB or 16MB, depending on the total size of the SGA. If the SGA is less than or equal to 128MB, a granule is 4MB; otherwise, it is 16MB.

In the next few subsections, we will cover the highlights of how Oracle uses each section in the SGA. More information on how to adjust the initialization parameters associated with these areas can be found in Chapter 8.
Buffer Caches

The database buffer cache holds blocks of data from disk that have been recently read to satisfy a `select` statement or that contain modified blocks that have been changed or added from a DML statement. As of Oracle9i, the memory area in the SGA that holds these data blocks is dynamic. This is a good thing, considering that there may be tablespaces in the database with block sizes other than the default block size; tablespaces with up to five different block sizes (one block size for the default, and up to four others) require their own buffer cache. As the processing and transactional needs change during the day or during the week, the values of `DB_CACHE_SIZE` and `DB_nK_CACHE_SIZE` can be dynamically changed without restarting the instance to enhance performance for a tablespace with a given block size.

Oracle can use two additional caches with the same block size as the default (DB_CACHE_SIZE) block size: the KEEP buffer pool and the RECYCLE buffer pool. As of Oracle9i, both of these pools allocate memory independently of other caches in the SGA.

When a table is created, you can specify the pool where the table's data blocks will reside by using the `BUFFER_POOL KEEP` or `BUFFER_POOL_RECYCLE` clause in the `STORAGE` clause. For tables that are used frequently throughout the day, it would be advantageous to place this table into the KEEP buffer pool to minimize the I/O needed to retrieve blocks in the table.

Shared Pool

The shared pool contains two major subcaches: the library cache and the data dictionary cache. The shared pool is sized by the `SHARED_POOL_SIZE` initialization parameter. This is another dynamic parameter that can be resized as long as the total SGA size is less than `SGA_MAX_SIZE` or `SGA_TARGET`.

Library Cache

The `library cache` holds information about SQL and PL/SQL statements that are run against the database. In the library cache, because it is shared by all users, many different database users can potentially share the same SQL statement.

Along with the SQL statement itself, the execution plan and parse tree of the SQL statement are stored in the library cache. The second time an identical SQL statement is run, by the same user or a different user, the execution plan and parse tree are already computed, improving the execution time of the query or DML statement.

If the library cache is sized too small, the execution plans and parse trees are flushed out of the cache, requiring frequent reloads of SQL statements into the library cache. See Chapter 9 for ways to monitor the efficiency of the library cache.

Data Dictionary Cache

The `data dictionary` is a collection of database tables, owned by the SYS and SYSTEM schemas, that contain the metadata about the database, its structures, and the privileges and roles of database users. The `data dictionary cache` holds cached blocks from the data dictionary. Data blocks from tables in the data dictionary are used continually to assist in processing user queries and other DML commands.

If the data dictionary cache is too small, requests for information from the data dictionary will cause extra I/O to occur; these I/O-bound data dictionary requests are called `recursive calls` and should be avoided by sizing the data dictionary cache correctly.

Redo Log Buffer

The `redo log buffer` holds the most recent changes to the data blocks in the datafiles. When the redo log buffer is one-third full, or every three seconds, redo log records are written to the redo
Chapter 1: Getting Started with the Oracle Architecture

log files. The entries in the redo log buffer, once written to the redo log files, are critical to database recovery if the instance crashes before the changed data blocks are written from the buffer cache to the datafiles. A user’s committed transaction is not considered complete until the redo log entries have been successfully written to the redo log files.

Large Pool
The large pool is an optional area of the SGA. It is used for transactions that interact with more than one database, message buffers for processes performing parallel queries, and RMAN parallel backup and restore operations. As the name implies, the large pool makes available large blocks of memory for operations that need to allocate large blocks of memory at a time.

The initialization parameter LARGE_POOL_SIZE controls the size of the large pool and is a dynamic parameter as of Oracle9i release 2.

Java Pool
The Java pool is used by the Oracle JVM (Java Virtual Machine) for all Java code and data within a user session. Storing Java code and data in the Java pool is analogous to SQL and PL/SQL code cached in the shared pool.

Streams Pool
New to Oracle 10g, the streams pool is sized by using the initialization parameter STREAMS_POOL_SIZE. The streams pool holds data and control structures to support the Oracle Streams feature of Oracle Enterprise Edition. Oracle Streams manages the sharing of data and events in a distributed environment. If the initialization parameter STREAMS_POOL_SIZE is uninitialized or set to zero, the memory used for Streams operations is allocated from the shared pool and may use up to 10 percent of the shared pool. For more information on Oracle Streams, see Chapter 18.

Program Global Area
The Program Global Area is an area of memory allocated and private for one process. The configuration of the PGA depends on the connection configuration of the Oracle database: either shared server or dedicated.

In a shared server configuration, multiple users share a connection to the database, minimizing memory usage on the server, but potentially affecting response time for user requests. In a shared server environment, the SGA holds the session information for a user instead of the PGA. Shared server environments are ideal for a large number of simultaneous connections to the database with infrequent or short-lived requests.

In a dedicated server environment, each user process gets its own connection to the database; the PGA contains the session memory for this configuration.

The PGA also includes a sort area. The sort area is used whenever a user request requires a sort, bitmap merge, or hash join operation.

As of Oracle9i, the PGA_AGGREGATE_TARGET parameter, in conjunction with the WORKAREA_SIZE_POLICY initialization parameter, can ease system administration by allowing the DBA to choose a total size for all work areas and let Oracle manage and allocate the memory between all user processes.

Software Code Area
Software code areas store the Oracle executable files that are running as part of an Oracle instance. These code areas are static in nature, and only change when a new release of the software is installed.
Typically, the Oracle software code areas are located in a privileged memory area separate from other user programs.

Oracle software code is strictly read-only and can be installed either shared or non-shared. Installing Oracle software code as sharable saves memory when multiple Oracle instances are running on the same server at the same software release level.

**Background Processes**

When an Oracle instance starts, multiple background processes start. A background process is a block of executable code designed to perform a specific task. Figure 1-6 shows the relationship between the background processes, the database, and the Oracle SGA. In contrast to a foreground

---

**FIGURE 1-6.** *Oracle background processes*
process, such as a SQL*Plus session or a web browser, a background process works behind the scenes. Together, the SGA and the background processes comprise an Oracle instance.

**SMON**
SMON is the *System Monitor* process. In the case of a system crash or instance failure, due to a power outage or CPU failure, the SMON process performs crash recovery by applying the entries in the online redo log files to the datafiles. In addition, temporary segments in all tablespaces are purged during system restart.

One of SMON’s routine tasks is to coalesce the free space in tablespaces on a regular basis if the tablespace is dictionary managed.

**PMON**
If a user connection is dropped, or a user process otherwise fails, PMON, also known as the *Process Monitor*, does the cleanup work. It cleans up the database buffer cache along with any other resources that the user connection was using. For example, a user session may be updating some rows in a table, placing a lock on one or more of the rows. A thunderstorm knocks out the power at the user’s desk, and the SQL*Plus session disappears when the workstation is powered off. Within moments, PMON will detect that the connection no longer exists and perform the following tasks:

- Roll back the transaction that was in progress when the power went out
- Mark the transaction’s blocks as available in the buffer cache
- Remove the locks on the affected rows in the table
- Remove the process ID of the disconnected process from the list of active processes

PMON will also interact with the listeners by providing information about the status of the instance for incoming connection requests.

**DBWn**
The *database writer* process, known as DBWR in older versions of Oracle, writes new or changed data blocks (known as *dirty blocks*) in the buffer cache to the datafiles. Using an LRU algorithm, DBWn writes the oldest, least active blocks first. As a result, the most commonly requested blocks, even if they are dirty blocks, are in memory.

Up to 20 DBWn processes can be started, DBW0 through DBW9 and DBWa through DBWj. The number of DBWn processes is controlled by the `DB_WRITER_PROCESSES` parameter.

**LGWR**
LGWR, or *Log Writer*, is in charge of redo log buffer management. LGWR is one of the most active processes in an instance with heavy DML activity. A transaction is not considered complete until LGWR successfully writes the redo information, including the commit record, to the redo log files. In addition, the dirty buffers in the buffer cache cannot be written to the datafiles by DBWn until LGWR has written the redo information.

If the redo log files are grouped, and one of the multiplexed redo log files in a group is damaged, LGWR writes to the remaining members of the group, and records an error in the alert log file. If all members of a group are unusable, the LGWR process fails and the entire instance hangs until the problem can be corrected.
If the database is in ARCHIVELOG mode, then the archiver process, or ARCn, copies redo logs to one or more destination directories, devices, or network locations whenever a redo log fills up and redo information starts to fill the next redo log in sequence. Optimally, the archive process finishes before the filled redo log is needed again; otherwise, serious performance problems occur—users cannot complete their transactions until the entries are written to the redo log files, and the redo log file is not ready to accept new entries because it is still being written to the archive location. There are at least three potential solutions to this problem: make the redo log files larger, increase the number of redo log groups, and increase the number of ARCn processes. Up to ten ARCn processes can be started for each instance by increasing the value of the LOG_ARCHIVE_MAX_PROCESSES initialization parameter.

The checkpoint process, or CKPT, helps to reduce the amount of time required for instance recovery. During a checkpoint, CKPT updates the header of the control file and the datafiles to reflect the last successful SCN (System Change Number). A checkpoint occurs automatically every time a redo log file switch occurs. The DBWn processes routinely write dirty buffers to advance the checkpoint from where instance recovery can begin, thus reducing the Mean Time to Recovery (MTTR).

The RECO, or recoverer process, handles failures of distributed transactions (that is, transactions that include changes to tables in more than one database). If a table in the CCTR database is changed along with a table in the WHSE database, and the network connection between the databases fails before the table in the WHSE database can be updated, RECO will roll back the failed transaction.

Backup/Recovery Overview
Oracle supports many different forms of backup and recovery. Some of them can be managed at the user level, such as export and import; most of them are strictly DBA-centric, such as online or offline backups and using operating system commands or the RMAN utility.

Details for configuring and using these backup and recovery methods can be found in Chapter 12 and also in Chapter 13.

Export/Import
The export command is a standalone utility on all Oracle hardware and software platforms, and it’s started by running the command exp at the operating system command-line prompt or through the Oracle Enterprise Manager console in a GUI environment. Export is considered a logical backup, because the underlying storage characteristics of the tables are not recorded, only the table metadata, user privileges, and table data. Depending on the task at hand, and whether you have DBA privileges or not, the exp command can either export all tables in the database, all the tables of one or more users, or a specific set of tables.

For restoring from a database export, the import command, started by running the command imp, takes a binary format file created by export and imports it into the database with the assumption that the users in the exported database tables exist in the database where the import command is performed.

One advantage to using export and import is that a database power user may be able to manage their own backups and recoveries, especially in a development environment. Also, a binary file
generated by export is typically readable across Oracle versions, making a transfer of a small set of tables from an older version to a newer version of Oracle fairly straightforward.

Export and import are inherently “point in time” backups and therefore are not the most robust backup and recovery solutions if the data is volatile.

In Oracle 10g, Oracle Data Pump takes import and export operations to a new performance level. Exports to an external data source can be up to two times faster, and an import operation can be up to 45 times faster because Data Pump Import uses direct path loading, unlike traditional import. In addition, an export from the source database can be simultaneously imported into the target database without an intermediate dump file, saving time and administrative effort. Oracle Data Pump is implemented using the DBMS_DATAPUMP package with the expdb and impdb commands, and includes numerous other manageability features, such as fine-grained object selection. More information on Oracle Data Pump is provided in Chapter 17.

**Offline Backups**

One of the ways to make a physical backup of the database is to perform an offline backup. To perform an offline backup, the database is shut down and all database-related files, including datafiles, control files, SPFILEs, password files, and so forth, are copied to a second location. Once the copy operation is complete, the database instance can be started.

Offline backups are similar to export backups because they are point-in-time backups and therefore of less value if up-to-the minute recovery of the database is required and the database is not in archivelog mode. Another downside to offline backups is the amount of downtime necessary to perform the backup; any multinational company that needs 24/7 database access will most likely not do offline backups very often.

**Online Backups**

If a database is in archivelog mode, it is possible to do online backups of the database. The database can be open and available to users even while the backup is in process. The procedure for doing online backups is as easy as placing a tablespace into a backup state by using the alter tablespace users begin backup command, backing up the datafiles in the tablespace with operating system commands, and then taking the tablespace out of the backup state with the alter tablespace users end backup command.

**RMAN**

The backup tool Recovery Manager, known more commonly as RMAN, has been around since Oracle8. RMAN provides many advantages over other forms of backup. It can perform incremental backups of only changed data blocks in between full database backups while the database remains online throughout the backup.

RMAN keeps track of the backups via one of two methods: through the control file of the database being backed up, or through a recovery catalog stored in another database. Using the target database’s control file for RMAN is easy, but it’s not the best solution for a robust enterprise backup methodology. Although a recovery catalog requires another database to store the metadata for the target database along with a record of all backups, it is well worth it when all the control files in the target database are lost due to a catastrophic failure. In addition, a recovery catalog retains historical backup information that may be overwritten in the target database’s control file if the value of CONTROL_FILE_RECORD_KEEP_TIME is set too low.

RMAN is discussed in detail in Chapter 13.
Security Capabilities
In the next few sections, we’ll give a brief overview of the different ways that the Oracle 10g Database controls and enforces security in a database. Account security based on user and schema objects was covered in the section on database objects; the other security topics are covered here.

An in-depth look at these and other security capabilities within Oracle is covered in Chapter 10.

Privileges and Roles
In an Oracle database, privileges control access to both the actions a user can perform and the objects in the database. Privileges that control access to actions in the database are called system privileges, whereas privileges that control access to data and other objects are called object privileges.

To make assignment and management of privileges easier for the DBA, a database role groups privileges together. To put it another way, a role is a named group of privileges. In addition, a role can itself have roles assigned to it.

Privileges and roles are granted and revoked with the grant and revoke commands. The user group PUBLIC is neither a user nor a role, nor can it be dropped; however, when privileges are granted to PUBLIC, they are granted to every user of the database, both present and future.

System Privileges
System privileges grant the right to perform a specific type of action in the database, such as creating users, altering tablespaces, or dropping any view. Here is an example of granting a system privilege:

```sql
grant DROP ANY TABLE to SCOTT WITH ADMIN OPTION;
```

The user SCOTT can drop anyone’s table in any schema. The with grant option clause allows SCOTT to grant his newly granted privilege to other users.

Object Privileges
Object privileges are granted on a specific object in the database. The most common object privileges are SELECT, UPDATE, DELETE, and INSERT for tables, EXECUTE for a PL/SQL stored object, and INDEX for granting index-creation privileges on a table. In the following example, the user RJB can perform any DML on the JOBS table owned by the HR schema:

```sql
grant SELECT, UPDATE, INSERT, DELETE on HR.JOBS to RJB;
```

Auditing
To audit access to objects in the database by users, you can set up an audit trail on a specified object or action by using the audit command. Both SQL statements and access to a particular database object can be audited; the success or failure of the action (or both) can be recorded in the audit trail table, SYS.AUD$.

For each audited operation, Oracle creates an audit record with the username, the type of operation that was performed, the object involved, and a timestamp. Various data dictionary views, such as DBA_AUDIT_TRAIL and DBA_FGA_AUDIT_TRAIL, make interpreting the results from the raw audit trail table SYS.AUD$ easier.
CAUTION
Excessive auditing on database objects can have an adverse effect on performance. Start out with basic auditing on key privileges and objects, and expand the auditing when the basic auditing has revealed a potential problem.

Fine-grained Auditing
The fine-grained auditing capability has been enhanced in Oracle 10g and takes auditing one step further: Standard auditing can detect when a `select` statement was executed on an `EMPLOYEE` table; fine-grained auditing will record an audit record containing specific columns accessed in the `EMPLOYEE` table, such as the `SALARY` column.

Fine-grained auditing is implemented using the DBMS_FGA package along with the data dictionary view DBA_FGA_AUDIT_TRAIL. The data dictionary view DBA_COMMON_AUDIT_TRAIL combines standard audit records in DBA_AUDIT_TRAIL with fine-grained audit records.

Virtual Private Database
The Virtual Private Database feature of Oracle, first introduced in Oracle8i, couples fine-grained access control with a secure application context. The security policies are attached to the data, and not to the application; this ensures that security rules are enforced regardless of how the data is accessed.

For example, a medical application context may return a predicate based on the patient identification number accessing the data; the returned predicate will be used in a WHERE clause to ensure that the data retrieved from the table is only the data associated with the patient.

Label Security
Oracle Label Security provides a “VPD Out-of-the-Box” solution to restrict access to rows in any table based on the label of the user requesting the access and the label on the row of the table itself. Oracle Label Security administrators do not need any special programming skills to assign security policy labels to users and rows in the table.

This highly granular approach to data security can, for example, allow a DBA at an Application Service Provider (ASP) to create only one instance of an accounts receivable application and to use Label Security to restrict rows in each table to an individual company’s accounts receivable information.

Real Application Clusters
Oracle’s Real Application Clusters (RAC) feature, known in previous Oracle versions as the Oracle Parallel Server option, allows more than one instance, on separate servers, to access the same database files.

A RAC installation can provide extreme high availability for both planned and unplanned outages. One instance can be restarted with new initialization parameters while the other instance is still servicing requests against the database. If one of the hardware servers crashes due to a fault of some type, the Oracle instance on the other server will continue to process transactions, even from users who were connected to the crashed server, transparently and with minimal downtime.
RAC, however, is not a software-only solution: The hardware that implements RAC has special requirements. The shared database should be on a RAID-enabled disk subsystem to ensure that each component of the storage system is fault tolerant. In addition, RAC requires a high-speed interconnect, or a private network, between the nodes in the cluster to support messaging and transfer of blocks from one instance to another using the Cache Fusion mechanism.

The diagram in Figure 1-7 shows a two-node RAC installation. How to set up and configure Real Application Clusters is discussed in depth in Chapter 11.

**Oracle Streams**

As a component of Oracle Enterprise Edition, *Oracle Streams* is the higher-level component of the Oracle infrastructure that complements Real Application Clusters. Oracle Streams allows the smooth flow and sharing of both data and events within the same database or from one database to another. It is another key piece in Oracle’s long list of high-availability solutions, tying together and enhancing Oracle’s message queuing, data replication, and event management functions. More information on how to implement Oracle Streams can be found in Chapter 18.

**Oracle Enterprise Manager**

Oracle Enterprise Manager (OEM) is a valuable set of tools that facilitates the comprehensive management of all components of an Oracle infrastructure, including Oracle database instances, Oracle application servers, and web servers. If a management agent exists for a third-party application, then OEM can manage the third-party application in the same framework as any Oracle-supplied target.

OEM is fully web-enabled via Netscape or Internet Explorer, and as a result any operating system platform that supports Netscape or IE can be used to launch the OEM console.
One of the key decisions to make when using OEM with Oracle Grid Control is the location to store the management repository. The OEM management repository is stored in a database separate from the nodes being managed or monitored. The metadata from the nodes and services is centralized and facilitates the administration of these nodes. The management repository database should be backed up often and separately from the databases being managed.

An installation of OEM provides a tremendous amount of value “out of the box.” When the OEM installation is complete, e-mail notifications are already set up to send messages to the SYSMAN or any other e-mail account for critical conditions, and the initial target discovery is automatically completed.

**Oracle Initialization Parameters**

An Oracle database uses initialization parameters to configure memory settings, disk locations, and so forth. There are two ways to store initialization parameters: using an editable text file and using a server-side binary file. Regardless of the method used to store the initialization parameters, there is a defined set of basic initialization parameters (as of Oracle 10g) that every DBA should be familiar with when creating a new database.

As of Oracle 10g, initialization parameters fall into two broad categories: basic initialization parameters and advanced initialization parameters. As Oracle becomes more and more self-managing, the number of parameters that a DBA must be familiar with and adjust on a daily basis is reduced.

**Basic Initialization Parameters**

The list of Oracle 10g basic initialization parameters appears in Table 1-3 along with a brief description of each. In the sections that follow, we will give some further explanation and advice.

<table>
<thead>
<tr>
<th>Initialization Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLUSTER_DATABASE</td>
<td>Enables this node to be a member of a cluster.</td>
</tr>
<tr>
<td>COMPATIBLE</td>
<td>Allows a new database version to be installed while ensuring compatibility with the release specified by this parameter.</td>
</tr>
<tr>
<td>CONTROL_FILES</td>
<td>Specifies the location of the control files for this instance.</td>
</tr>
<tr>
<td>DB_BLOCK_SIZE</td>
<td>Specifies the size of Oracle blocks. This block size is used for the SYSTEM, SYSAUX, and temporary tablespaces at database creation.</td>
</tr>
<tr>
<td>DB_CREATE_FILE_DEST</td>
<td>The default location for OMF datafiles. Also specifies the location of control files and redo log files if DB_CREATE_ONLINE_LOG_DEST_n is not set.</td>
</tr>
<tr>
<td>DB_CREATE_ONLINE_LOG_DEST_n</td>
<td>The default location for OMF control files and online redo log files.</td>
</tr>
<tr>
<td>DB_DOMAIN</td>
<td>The logical domain name where the database resides in a distributed database system (for example, us.oracle.com).</td>
</tr>
<tr>
<td>DB_NAME</td>
<td>A database identifier of up to eight characters. Preppended to the DB_DOMAIN value for a fully qualified name (for example, marketing.us.oracle.com).</td>
</tr>
</tbody>
</table>

**TABLE 1-3. Basic Initialization Parameters**
### Initialization Parameter Description

<table>
<thead>
<tr>
<th>Initialization Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DB_RECOVERY_FILE_DEST</td>
<td>The default location for the recovery area. Must be set along with DB_RECOVERY_FILE_DEST_SIZE.</td>
</tr>
<tr>
<td>DB_RECOVERY_FILE_DEST_SIZE</td>
<td>The maximum size, in bytes, for the files used for recovery in the recovery area location.</td>
</tr>
<tr>
<td>INSTANCE_NUMBER</td>
<td>In a RAC installation, the instance number of this node in the cluster.</td>
</tr>
<tr>
<td>JOB_QUEUE_PROCESSES</td>
<td>The maximum number of processes allowed for executing jobs, ranging from 0 to 1000.</td>
</tr>
<tr>
<td>LOG_ARCHIVE_DEST_n</td>
<td>For ARCHIVELOG mode, up to ten locations for sending archived log files.</td>
</tr>
<tr>
<td>LOG_ARCHIVE_DEST_STATE_n</td>
<td>Sets the availability of the corresponding LOG_ARCHIVE_DEST_n sites.</td>
</tr>
<tr>
<td>NLS_LANGUAGE</td>
<td>Specifies the default language of the database, including messages, day and month names, and sorting rules (for example, ‘AMERICAN’).</td>
</tr>
<tr>
<td>NLS_TERRITORY</td>
<td>The territory name used for day and week numbering (for example, ‘SWEDEN’, ‘TURKEY’, or ‘AMERICA’).</td>
</tr>
<tr>
<td>OPEN_CURSORS</td>
<td>The maximum number of open cursors per session.</td>
</tr>
<tr>
<td>PGA_AGGREGATE_TARGET</td>
<td>The total memory to allocate for all server processes in this instance.</td>
</tr>
<tr>
<td>PROCESSES</td>
<td>The maximum number of operating system processes that can connect to Oracle simultaneously. SESSIONS and TRANSACTIONS are derived from this value.</td>
</tr>
<tr>
<td>REMOTE_LISTENER</td>
<td>A network name resolving to an Oracle Net remote listener.</td>
</tr>
<tr>
<td>REMOTE_LOGIN_PASSWORDFILE</td>
<td>Specifies how Oracle uses password files. Required for RAC.</td>
</tr>
<tr>
<td>ROLLBACK_SEGMENTS</td>
<td>Names of private rollback segments to bring online, if undo management is not used for transaction rollback.</td>
</tr>
<tr>
<td>SESSIONS</td>
<td>The maximum number of sessions, and therefore simultaneous users, in the instance. Defaults to 1.1*PROCESSES + 5.</td>
</tr>
<tr>
<td>SGA_TARGET</td>
<td>Specifies the total size of all SGA components; this parameter automatically determines DB_CACHE_SIZE, SHARED_POOL_SIZE, LARGE_POOL_SIZE, and JAVA_POOL_SIZE.</td>
</tr>
<tr>
<td>SHARED_SERVERS</td>
<td>The number of shared server processes to allocate when an instance is started.</td>
</tr>
<tr>
<td>STAR_TRANSFORMATION_ENABLED</td>
<td>Controls query optimization when star queries are executed.</td>
</tr>
<tr>
<td>UNDO_MANAGEMENT</td>
<td>Specifies whether undo management is automatic (AUTO) or manual (MANUAL). If MANUAL is specified, rollback segments are used for undo management.</td>
</tr>
<tr>
<td>UNDO_TABLESPACE</td>
<td>The tablespace to use when UNDO_MANAGEMENT is set to AUTO.</td>
</tr>
</tbody>
</table>

**TABLE 1-3.** Basic Initialization Parameters (continued)
regarding how some of these parameters should be set, depending on the hardware and software environment, the types of applications, and the number of users in the database.

Some of these parameters will be revisited in the section titled “Software Installation,” where we will set the initial parameters for the SGA, file locations, and other limits.

**COMPATIBLE**

The COMPATIBLE parameter allows a newer version of Oracle to be installed while restricting the feature set of the new version as if an older version of Oracle was installed. This is a good way to move forward with a database upgrade while remaining compatible with an application that may fail when it runs with the new version of the software. The COMPATIBLE parameter can then be bumped up as the applications are reworked or rewritten to work with the new version of the database.

The downside of using this parameter is that none of the new applications for the database can take advantage of new features until the COMPATIBLE parameter is set to the same value as the current release.

**DB_NAME**

DB_NAME specifies the local portion of the database name. It can be up to eight characters and must begin with an alphanumeric character. Once set, it cannot be changed without re-creating the database; the DB_NAME is recorded in each datafile, redo log file, and control file in the database. At database startup, the value of this parameter must match the value of DB_NAME recorded in the control file.

**DB_DOMAIN**

DB_DOMAIN specifies the name of the network domain where the database will reside. The combination of DB_NAME and DB_DOMAIN must be unique within a distributed database system.

**DB_RECOVERY_FILE_DEST and DB_RECOVERY_FILE_DEST_SIZE**

When database recovery operations occur, either due to an instance failure or a media failure, it is convenient to have a flash recovery area to store and manage files related to a recovery or backup operation. Starting with Oracle 10g, the parameter DB_RECOVERY_FILE_DEST can be a directory location on the local server, a network directory location, or an ASM (Automatic Storage Management) disk area. The parameter DB_RECOVERY_FILE_DEST_SIZE places a limit on how much space is allowed for the recovery or backup files.

These parameters are optional, but if specified, Recovery Manager (RMAN) can automatically manage the files needed for backup and recovery operations. The size of this recovery area should be large enough to hold two copies of all datafiles, incremental RMAN backups, online redo logs, archived log files not yet backed up to tape, the SPFILE, and the control file.

**CONTROL_FILES**

The CONTROL_FILES parameter is not required. If it is not specified, Oracle creates one control file in a default location, or if OMF is configured, in the location specified by either DB_CREATE_FILE_DEST or DB_CREATE_ONLINE_LOG_DEST_n.

However, it is strongly recommended that multiple copies of the control file be created on separate physical volumes. Control files are so critical to the database integrity and are so small that at least three multiplexed copies of the control file should be created on separate physical disks. In addition, the command *alter database backup controlfile to trace* should be executed to create a text-format copy of the control file in the event of a major disaster.
The following example specifies three locations for copies of the control file:

```
CONTROL_FILES = (/u01/oracle10g/test/control01.ctl,
                   /u03/oracle10g/test/control02.ctl,
                   /u07/oracle10g/test/control03.ctl)
```

**DB_BLOCK_SIZE**

The parameter `DB_BLOCK_SIZE` specifies the size of the default Oracle block in the database. At database creation, the SYSTEM, TEMP, and SYSAUX tablespaces are created with this block size. Ideally, this parameter is the same as or a multiple of the operating system block size for I/O efficiency.

Before Oracle9i, you might specify a smaller block size (4KB or 8KB) for OLTP systems and a larger block size (up to 32KB) for DSS systems. However, now that tablespaces with up to five block sizes can coexist in the same database, a smaller value for `DB_BLOCK_SIZE` is fine. However, 8KB is probably preferable as a minimum for any database, unless it has been rigorously proven in the target environment that a 4KB block size will not cause performance issues.

**SGA_TARGET**

Another way that Oracle 10g can facilitate a “set it and forget it” database is by the ability to specify a total amount of memory for all SGA components. If `SGA_TARGET` is specified, the parameters `DB_CACHE_SIZE`, `SHARED_POOL_SIZE`, `LARGE_POOL_SIZE`, and `JAVA_POOL_SIZE` are automatically sized by Automatic Shared Memory Management (ASMM). If any of these four parameters are manually sized when `SGA_TARGET` is also set, ASMM uses the manually sized parameters as minimums.

Once the instance starts, the automatically sized parameters can by dynamically increased or decreased, as long as the parameter `SGA_MAX_SIZE` is not exceeded. The parameter `SGA_MAX_SIZE` specifies a hard upper limit for the entire SGA, and it cannot be exceeded or changed until the instance is restarted.

Regardless of how the SGA is sized, be sure that enough free physical memory is available in the server to hold the components of the SGA and all background processes; otherwise, excessive paging will occur and performance will suffer.

**DB_CACHE_SIZE and DB_nK_CACHE_SIZE**

The parameter `DB_CACHE_SIZE` specifies the size of the area in the SGA to hold blocks of the default size, including those from the SYSTEM, TEMP, and SYSAUX tablespaces. Up to four other caches can be defined if there are tablespaces with block sizes other than the SYSTEM and SYSAUX tablespaces. The value of `n` can be 2, 4, 8, 16, or 32; if the value of `n` is the same as the default block size, the corresponding `DB_nK_CACHE_SIZE` parameter is illegal. Although this parameter is not one of the basic initialization parameters, it becomes very basic when you transport a tablespace from another database with a block size other than `DB_BLOCK_SIZE`!

There are distinct advantages to a database containing multiple block sizes. The tablespace handling OLTP applications can have a smaller block size, and the tablespace with the data warehouse table can have a larger block size. However, be careful when allocating memory for each of these cache sizes so as not to allocate too much memory for one at the expense of another. As of Oracle9i, Oracle’s Buffer Cache Advisory feature monitors the cache usage for each cache
size in the view V$DB_CACHE_ADVICE to assist the DBA in sizing these memory areas. More information on how to use the Buffer Cache Advisory feature can be found in Chapter 8.

**SHARED_POOL_SIZE, LARGE_POOL_SIZE, and JAVA_POOL_SIZE**
The parameters SHARED_POOL_SIZE, LARGE_POOL_SIZE, and JAVA_POOL_SIZE, which size the shared pool, large pool, and Java pool, respectively, are automatically sized by Oracle if the SGA_TARGET initialization parameter is specified. More information on manually tuning these areas can be found in Chapter 8.

**PROCESSES**
The value for the PROCESSES initialization parameter represents the total number of processes that can simultaneously connect to the database. This includes both the background processes and user processes; a good starting point for the PROCESSES parameter would be 15 for the background processes plus the number of expected maximum concurrent users; for a smaller database, 50 is a good starting point, because there is little or no overhead associated with making PROCESSES too big.

**UNDO_MANAGEMENT and UNDO_TABLESPACE**
Automatic Undo Management, introduced in Oracle9i, eliminates or at least greatly reduces the headaches in trying to allocate the right number and size of rollback segments to handle the undo information for transactions. Instead, a single undo tablespace is specified for all undo operations (except for a SYSTEM rollback segment), and all undo management is handled automatically when the UNDO_MANAGEMENT parameter is set to AUTO.

The remaining task for the DBA is sizing the undo tablespace. Data dictionary views such as V$UNDOSTAT and the Undo Advisor can help the DBA adjust the size of the undo tablespace. Multiple undo tablespaces may be created; for example, a smaller undo tablespace is online during the day to handle relatively small transaction volumes, and a larger undo tablespace is brought online overnight to handle batch jobs and long-running queries that load the data warehouse and need transactional consistency. Only one undo tablespace may be active at any given time.

**Advanced Initialization Parameters**
The advanced initialization parameters include the balance of the initialization parameters not listed here, for a total of 255 of them in Release 1 of Oracle Database 10g. Most of these can be automatically set and tuned by the Oracle instance when the basic initialization parameters are set. We will review some of these in the next section, titled “Software Installation.”

**Software Installation**
Although Oracle’s installation software becomes easier and easier to use, it is very tempting to open the box of CDs and start the installation right away. Although this is fine if you’re going to experiment with some new database features, a lot more planning is required to perform a successful installation without a lot of rework or even reinstallation a month from now. Although the complete details of an Oracle Database 10g installation is beyond the scope of this book, we will cover the basics of an Oracle install using the Oracle Universal Installer (OUI) along with a basic template for doing a manual install of the database using the create database command.
any case, a thorough review of the installation guide for your specific platform is another key to a successful Oracle database deployment. Here is a checklist of issues that should be addressed or resolved before starting the installation:

■ Decide on the local database name and which domain will contain this database. These names are set in the initialization parameters DB_NAME and DB_DOMAIN.

■ For the first project to use the database, estimate the number of tables and indexes, as well as their size, to plan for disk space estimates beyond what is required for the Oracle SYSTEM tablespace and the associated Oracle software and tools.

■ Plan the locations of the physical datafiles on the server’s disk to maximize performance and recoverability. The more physical disks, the better. If a RAID or network attached storage (NAS) area will be used for the datafiles, then consider OMF to manage the placement of the datafiles.

■ Review and understand the basic initialization parameters, and plan on using an SPFILE before going live, if not using an SPFILE right away.

■ Select the database character set, along with an alternate character set. Although it’s easy to let the character sets default on install, you may need to consider where the users of the database are located and their language requirements. Character sets can be changed after installation only if the new character set is a superset of the existing character set.

■ Decide on the best default database block size. The default block size defined by DB_BLOCK_SIZE cannot be changed later without reinstalling the database. Although this decision is not critical to future expansion of the database because Oracle can support tablespaces with multiple block sizes, having an incorrectly sized default block size can reduce the performance of operations using the SYSTEM, TEMP, or SYSAUX tablespace.

■ Make sure that all non-administrative users are assigned a non-SYSTEM tablespace as their default tablespace. No user objects should ever be stored in the SYSTEM tablespace.

■ Automatic Undo Management is a must to ease the administration of transaction undo information. The extra space needed in the undo tablespace is well worth the investment in productivity for both the DBA and users.

■ Plan a backup and recovery strategy. Decide how often the database needs to be backed up; use more than one method to back up the database. One of the key questions to ask when selecting a backup strategy is, How long can we afford to have our database down? If the database only processes batch jobs at night, then a full backup every week and daily incremental backups are probably fine. If you’re running a website in front of your database that sells widgets to customers all over the world, 24/7, and the cost of being down is six figures a minute, then the investment in an Oracle Real Applications Cluster (RAC) environment along with Data Guard for disaster recovery is probably cost-justified.

Familiarity with a couple key websites is a must. Oracle Technology Network (OTN), at http://otn.oracle.com, has a wealth of information, including white papers, free tools, sample code, and the online version of Oracle Magazine. There is no charge for using OTN, other than registering on the site.
Purchasing a license for Oracle database software is a good start, but an Oracle support contract with web support is the key to a successful installation and deployment. Using Oracle’s Metalink (http://metalink.oracle.com) means you might never have to leave the friendly confines of your web browser to keep your database up and running. Through Metalink, you can submit a support request, search through other support requests, download patches, download white papers, and search the bug database.

A successful initial software installation is the first step. The database environment grows and evolves with every new business requirement and application that comes your way. For more information on how to successfully plan the implementation of a large database development project, see Chapter 5.

Overview of Licensing and Installation Options
Regardless of the software and hardware platform you’re installing Oracle on, the types of installations you can perform are the same:

- **Enterprise Edition** This is the most feature-laden and extensible version of the Oracle database. It includes features such as Flashback Database and allows you to add additional pieces of licensed functionality, such as Oracle Spatial, Real Application Clusters, Oracle OLAP, Oracle Label Security, and Oracle Data Mining.

- **Standard Edition** This edition provides a good subset of the features of the Enterprise Edition, including Real Application Clusters for up to four total CPUs, but the additional add-on pieces, such as Oracle Label Security, cannot be added to the Standard Edition.

- **Standard Edition One** This edition provides the same features of Standard Edition, except for Real Application Clusters, and is limited to a single server with a maximum of two CPUs.

- **Personal Edition** Allows for the development of applications that will run on either the Standard or Enterprise Edition. This edition cannot be used in a production environment.

As of Oracle 10g, licensing for the Oracle Database is only by named user or CPU, and there is no longer a concurrent user licensing option. Therefore, the DBA should use the initialization parameter LICENSE_MAX_USERS to specify the maximum number of users that can be created in the database. As a result, LICENSE_MAX_SESSIONS and LICENSE_SESSIONS_WARNING are deprecated in Oracle 10g.

Using OUI to Install the Oracle Software
The Oracle Universal Installer (OUI) is used to install and manage all Oracle components for both the server-side and client-side components. You can also deinstall any Oracle products from the initial OUI screens.

During the server installation, you will choose the version of Oracle Database 10g from the list in the previous section: Enterprise Edition, Standard Edition, or Personal Edition.

It is strongly recommended that you create a starter database when prompted during the install. Creating the starter database is a good way to make sure the server environment is set up correctly, as well as to review any new features of Oracle 10g. The starter database may also be a good candidate as a Grid Control repository for either Enterprise Manager or Recovery Manager.
At some point in the software installation, the Database Configuration Assistant (DBCA) takes over and prompts you for the parameters necessary to size and configure your database. The installation steps in the next session assume you have already completed the software installation and have created a starter database; we will create and configure a second database on the same server with the DBCA.

**NOTE**

As of Oracle 10g, the DBCA can configure nodes in a Real Application Clusters environment.

### Using the DBCA to Create a Database

At the operating system command prompt, launch the DBCA by typing `dbca`. In the subsections that follow, we will provide additional tips and guidance for most of the screens during the creation of the database.

#### DBCA Options

After an initial welcome screen, you are presented with a choice of four options:

- **Create a Database** This one is fairly straightforward; you are creating a new database from scratch, using a template as a starting point.
- **Configure Database Options in a Database** Some of the system parameters for an existing database installation can be changed, such as changing from a dedicated server mode to shared server.
- **Delete a Database** This one is also straightforward, and very dangerous! It will shut down the database and delete all the datafiles and control files associated with the database. You will need the SYS or SYSTEM password to proceed with this option if you are not using operating system authentication.
- **Manage Templates** This option allows you to add, modify, or delete templates. During a DBCA session, once all database parameters have been gathered, you have the option to save your settings as a template. In many cases, the predefined templates that Oracle provides are not quite perfect for your environment, and it is a timesaver to be able to save your database options for selection as a template in a future DBCA session.

#### Selecting a Database Template

In Figure 1-8, you are presented with the list of templates available. If you have created your own templates in previous DBCA sessions, they will appear on this screen also.

The template choices are as follows:

- **Custom Database** Use this option if you have performed many installations and know ahead of time the values for all the options you need in the database. This option is good if you are creating a new template from scratch or have very specific requirements for the configuration of your database.
Data Warehouse  This template is for database environments where users are performing numerous, complex queries that join many large tables for reporting, forecasting, and analytics.

General Purpose  If you are not sure of the intended use of your database yet, or if you need to host users with both analytical and transaction processing requirements, choose this template.

Transaction Processing  In 24/7 environments where the number of users is high, the transactions are heavy, but short, and the bulk of the activity is creating and updating, use this template.

In this installation, we are choosing the General Purpose template. It combines the features of both a data warehouse and an OLTP environment into a single database; use this option if you must use this database for both environments. Ideally, however, any database you create should be configured and tuned for the types of users and transactions on the database.

FIGURE 1-8. Database template selection
Database Identification

In step 3 of the DBCA, you will identify the name of the instance along with the global database name; if you type the fully qualified database name, it will use all characters up to the first period as the instance name, or Oracle System Identifier (SID). Figure 1-9 shows the Database Identification screen.

Note the distinction between a SID, an instance name, and a database. You may have one or more databases on a server; each database may have one or more instances opening each database. The instance names that use the same database must be unique. On the server, if there is more than one instance with the same name (but opening different databases), the SID associated with the instance must be unique.

TIP

If the global database name needs to be changed in the future, you must use the alter database command to change it, in addition to changing it in the initialization parameter file. The global database name is stored in the data dictionary when the database is created.

FIGURE 1-9. Database Identification screen
Unless you have an existing domain, use the default domain name .world. Check with your system administrator to see if a specific global database name should be used.

**Management Options**
The next screen, shown in Figure 1-10, specifies the database management options available. If you want to set up web services to control this database with the web-enabled Oracle Enterprise Manager, check the first box. If there is a Grid Control management service running on the network (known as an *Enterprise Manager Repository agent* in previous Oracle versions), you have the option to specify which management service will control this database.

If there is no Grid Control management service available, you can manage the database locally with Database Control. Within the Database Control choice, you can specify where alerts, warnings, and other database notifications are sent by specifying an e-mail address and the mail server name that will forward the e-mail messages. If you want to set up automated daily backups, you can specify the time of day for the backups.

In our example, we don’t have any Grid Control management service agents available, so we choose Database Control to manage our Enterprise Manager interface. We also specify the e-mail address where we want our server alerts to be sent (in this case, bob@rjbdba.com).

**FIGURE 1-10. Database Management Options screen**
Database Credentials
In Figure 1-11, you provide the initial passwords for the administrative user accounts. After the installation, be sure to create at least one account with DBA privileges to use instead of SYS or SYSTEM for day-to-day administrative tasks.

If you wish to use different passwords for the SYS, SYSTEM, DBSNMP, and SYSMAN accounts, you can do that on this screen; later in the installation, after the database has been created, you will have the opportunity to change the passwords for all of the approximately 30 user accounts created in a typical installation.

Storage Options
The database can use a number of different methods for storing datafiles, control files, and redo log files. If you have the resources to dedicate another database instance for managing disk space, choose ASM. An instance to manage ASM is fairly lightweight; typically, it will use no more than 64MB.
of RAM. If you are in a Real Application Clusters environment and you don’t have a cluster file system available (such as OCFS), then choose Raw Devices. Figure 1-12 shows these options on the Storage Options screen.

File Locations
The next screen, shown in Figure 1-13, is where you select the locations for datafiles, control files, and redo log files, as well as the archiving and backup and recovery locations. You can use the locations provided with the template, provide your own single location for all files, or configure OMF (Oracle Managed Files) for this database. In all cases, you will have the option to fine-tune these locations in a later step.

New to Oracle 10g is the concept of a Flash Recovery Area. This is a dedicated location on disk, separate from the location of the database’s operational files, containing the backup files from RMAN. It is highly recommended that you use a Flash Recovery Area so that RMAN can more...
easily manage backup and recovery operations. As mentioned previously in this chapter, be sure that the Flash Recovery Area is large enough to hold at least two copies of all datafiles, incremental backups, control files, SPFILEs, and archived redo log files that are still on disk. Figure 1-14 shows the screen where you specify the location of the Flash Recovery Area, along with its default size.

You can also enable `archivelog` mode, as well as specify the location or locations for the archived redo log files. It is recommended that you leave archiving off until the database is installed, because this will increase the database creation time. The parameters for `archivelog` mode can easily be changed in init.ora or the SPFILE immediately after the database is up and running.
Database Components
In step 9 of the DBCA session, you are asked about installing sample schemas. It is highly recommended that you install the sample schemas; many tutorials and study guides rely on the sample schemas being in the database. They are also useful in that the samples demonstrate nearly all datatypes and constructs available in the database, ranging from bitmapped indexes to clustered tables and object types. The screen shown in Figure 1-15 allows you to specify the installation of the sample schemas. The second tab on this screen gives you the option to specify other scripts that you need to run against this database once it is created, such as scripts to create tablespaces for existing applications, special user accounts, and so forth.
NOTE
Sample schemas should not be installed in a production database for security and performance reasons.

If you choose to not install the sample schemas at this time, they can be created using the scripts in the directory $ORACLE_HOME/demo/schema after the database has been created.

Initialization Parameters
The screens in Figures 1-16 through 1-19 allow the DBA to adjust the key initialization parameters for the database. In previous sections of this chapter, we described many of the basic initialization
parameters that a DBA needs to understand. In the next few paragraphs, we will link those parameters with the values you enter on the screens.

Figure 1-16 shows the Memory tab of the Initialization Parameters screen. If you select Typical or select Custom with Shared Memory Management Automatic, Oracle will make assumptions about the memory it can use for the SGA and background processes. Even by defaulting many of the parameters in a Typical configuration, you can still specify how much of the server’s physical memory should be used for Oracle, depending on how much memory is used by the operating system, and whether any other applications are going to be running on this server along with Oracle.
Because Java Pool and the Large Pool are optional, you can set these to zero and give more memory to Shared Pool or Buffer Cache; however, even if you only have one Java Stored Procedure in the database, the value for Java Pool must be at least the size of one granule in the database—either 4MB or 16MB, but at least 20MB is recommended.

Figure 1-17 allows you to specify the default block size of the database, as well as the total number of processes that will be simultaneously connecting to Oracle, corresponding to the PROCESSES initialization parameter.

The Block Size parameter should be fine at either 4KB or preferably 8KB, as long as it’s a multiple of the operating system block size. Making sure that the database block size is correct is not as critical as it used to be in previous versions of Oracle because other tablespaces in the database can have block sizes different from the default block size.

FIGURE 1-17. Initialization Parameters, Sizing tab
Chapter 1: Getting Started with the Oracle Architecture

If the Oracle Server will not be running in Shared Server mode, and therefore every user connection will be a dedicated connection, be sure to make the Processes setting high enough to include all expected database connections plus at least 15 for background processes.

The next screen in the DBCA specifies the language parameters for the database. Unless you have a multinational organization and have special language requirements for the database end users, the default Database Character Set and National Character Set selections, based on your server settings, should be fine. Figure 1-18 shows the NLS-related parameters.

The next screen allows you to specify how users will connect to the database—via dedicated connections to the database or through a Shared Server environment. If you select Shared Server mode, you can edit some of the specifics of the Shared Server environment, such as what protocols
are used and how many connections are assigned per dispatcher. In Figure 1-19, we are selecting Shared Server mode, with two Shared Server processes available to handle the connection requests from the dispatchers.

While you are still on the Initialization Parameters screen, you are able to fine-tune the values of initialization parameters by clicking the All Initialization Parameters button. Figure 1-20 shows

![Database Configuration Assistant, Step 10 of 12: Initialization Parameters](image)

**FIGURE 1-19.** Initialization Parameters, Connection Mode tab
how you can modify the value for SGA_TARGET or other parameters, even though you may have already implicitly defined these parameters in previous screens.

**Database Storage**

On the DBCA Database Storage screen, you can review and revise the locations of the control files, datafiles, and redo log files, as well as multiplex the control files and create redo log file groups.
Figure 1-21 shows how we can multiplex the control files to three different disk locations. The names and locations of the control files on this screen determine the value of CONTROL_FILES in the initialization parameter file or SPFILE.

FIGURE 1-21. Database Storage screen
Chapter 1: Getting Started with the Oracle Architecture

Creation Options
In Figure 1-22, we are ready to create the database. In addition, we can use the information we provided in the previous screens and save it to a template. If in doubt, save it as a template; the

![Database Configuration Assistant](image)

**FIGURE 1-22.** Database and template creation options
storage required to save a template is minimal, and it can easily be deleted later by rerunning the DBCA.

Before the database is created, an HTML summary of your template is presented, as shown in Figure 1-23, and you have the option to save this report as an HTML file for documentation purposes.
Completing the Installation
After clicking OK on the HTML Summary screen, the DBCA performs the tasks to create the database and start the instance. A standard set of scripts is run when the database first starts; this includes the scripts that create the sample schemas, plus any custom scripts you may have specified earlier. Figure 1-24 shows the DBCA performing the initialization tasks.

Once the initialization and creation scripts have completed, a summary screen is presented, as in Figure 1-25, giving the location of the log file for this installation. It is recommended that you review this log file to ensure that there were no unexpected errors during the install. You should also save this log file with the other documentation for this database; it can also be useful for future installations as a baseline.

The Oracle instance you just created is up and running. You have the option to unlock other accounts created during this install and to assign passwords to them.

**FIGURE 1-24. Creating and starting the Oracle instance**
Manually Creating a Database

In some situations, it is preferable to manually create a database instead of using the DBCA. For example, a DBA may have to create the same database on 20 different servers, or the database may need some parameters in the `create database` command that aren't adjustable using the DBCA. Oracle provides a sample database-creation script that can be customized for a manual install. Alternatively, the DBA can use the DBCA to save a database script to a file, which can be edited later and run at the SQL*Plus command line.

Here are the basic steps needed to create a database manually. Some of these steps are operating system or platform dependent, and we will note the differences. Be sure to review the installation guide for your specific platform before attempting a manual installation. For example, under Windows, you will need to run the utility oradim.exe to create the Oracle background process and to set the relevant Registry values.

1. Decide on a directory structure for the database; it is recommended that you comply with Oracle's Optimal Flexible Architecture (OFA) standards when placing your files on disk. See Chapters 3 and 4 for more information on OFA.

2. Select an Oracle SID to distinguish this instance from any other ones running on this server. Frequently this is the same as the database name specified in the DB_NAME initialization parameter. At a Windows command prompt, you will type the following:
   ```sql
   set ORACLE_SID=rjdbdb
   ```
Under Unix, you will use either
export ORACLE_SID=rjbdb
or
setenv ORACLE_SID=rjbdb
depending on your default command shell.

3. Establish an authentication method for connecting privileged users to the database. Use
the orapwd command-line utility to create a password file if you want Oracle to authenticate
the privileged users, and you will set the initialization parameter REMOTE_LOGIN_PASSWORDFILE to EXCLUSIVE. If you are using operating system authentication, there
is no need for a password file; set REMOTE_LOGIN_PASSWORDFILE to NONE.

4. Create an initialization parameter file and place it in the default location for your platform,
at least initially for the install. Under Unix, the default location is $ORACLE_HOME/dbs;
under Windows it is $ORACLE_HOME\database. Here is a sample Oracle-supplied
initialization file:

```
# Cache and I/O
DB_BLOCK_SIZE=4096
DB_CACHE_SIZE=20971520

# Cursors and Library Cache
CURSOR_SHARING=SIMILAR
OPEN_CURSORS=300

# Diagnostics and Statistics
BACKGROUND_DUMP_DEST=/u01/oracle10g/admin/rjbdb/bdump
CORE_DUMP_DEST=/u01/oracle10g/admin/rjbdb/cdump
TIMED_STATISTICS=TRUE
USER_DUMP_DEST=/u01/oracle10g/admin/rjbdb/udump

# Control File Configuration
CONTROL_FILES=(/u01/oracle10g/prod/rjbdb/control01.ctl,
                /u02/oracle10g/prod/rjbdb/control02.ctl,
                /u03/oracle10g/prod/rjbdb/control03.ctl)

# Archive
LOG_ARCHIVE_DEST_1=LOCATION=/u06/oracle10g/oradata/rjbdb/archive
# New log archive format. If compatibility 10.0 and up,
# this is enforced.
LOG_ARCHIVE_FORMAT=%t_%s_%r.dbf
# The following parameter is deprecated in 10iR1
# LOG_ARCHIVE_START=TRUE
```
# Shared Server
# Starts shared server if set > 0.
SHARED_SERVERS=2
# Uncomment and use first DISPATCHERS parameter
# below when your listener is
# configured for SSL
# (listener.ora and sqlnet.ora)
# DISPATCHERS = "(PROTOCOL=TCPS) (SER=MODOSE)",
# "(PROTOCOL=TCPS) (PRE=oracle.aurora.server.SGiopServer)"
# DISPATCHERS="(PROTOCOL=TCP) (SER=MODOSE)",
# "(PROTOCOL=TCP) (PRE=oracle.aurora.server.SGiopServer)",
# "(PROTOCOL=TCP)"

# Miscellaneous
COMPATIBLE=10.0.0
DB_NAME=rjbdb

# Distributed, Replication and Snapshot
DB_DOMAIN=rjdba.com
REMOTE_LOGIN_PASSWORDFILE=EXCLUSIVE

# Network Registration
INSTANCE_NAME=rjbdb

# Pools
JAVA_POOL_SIZE=31457280
LARGE_POOL_SIZE=1048576
SHARED_POOL_SIZE=52428800

# Processes and Sessions
PROCESSES=150

# Redo Log and Recovery
FAST_START_MTTR_TARGET=300

# Resource Manager
RESOURCE_MANAGER_PLAN=SYSTEM_PLAN

# Sort, Hash Joins, Bitmap Indexes
SORT_AREA_SIZE=524288

# Automatic Undo Management
UNDO_MANAGEMENT=AUTO
UNDO_TABLESPACE=undotbs
5. Connect to the instance using SQL*Plus, as shown here:

   sqlplus /nolog
   connect SYS/password as sysdba

   Note that while the instance itself exists, there is not much that we can do with it because we have not started the database yet.

6. Create a server parameter file (SPFILE). If the initialization file is in the default location, this command will create the SPFILE:

   create spfile from pfile;

7. Start the instance using the following command:

   startup nomount

   Note that because we do not have a database created yet, this is the only option we can use with the startup command.

8. Issue the create database statement. Here is an example:

   CREATE DATABASE rjbdb
   
   USER SYS IDENTIFIED BY paris703
   USER SYSTEM IDENTIFIED BY tyler12
   LOGFILE GROUP 1 ('/u02/oracle10g/oradata/rjbdb/redo01.log') SIZE 100M,
   GROUP 2 ('/u04/oracle10g/oradata/rjbdb/redo02.log') SIZE 100M,
   GROUP 3 ('/u06/oracle10g/oradata/rjbdb/redo03.log') SIZE 100M
   MAXLOGFILES 5
   MAXLOGMEMBERS 5
   MAXLOGHISTORY 1
   MAXDATAFILES 100
   MAXINSTANCES 1
   CHARACTER SET US7ASCII
   NATIONAL CHARACTER SET AL16UTF16
   DATAFILE '/u01/oracle10g/oradata/rjbdb/system01.dbf' SIZE 325M REUSE
   EXTENT MANAGEMENT LOCAL
   SYSAUX DATAFILE '/u01/oracle10g/oradata/rjbdb/sysaux01.dbf' SIZE 325M REUSE
   DEFAULT TABLESPACE tbs_1
   DEFAULT TEMPORARY TABLESPACE temps1
   TEMPFILE '/u01/oracle10g/oradata/rjbdb/temp01.dbf'
   SIZE 20M REUSE
   UNDO TABLESPACE undotbs
   DATAFILE '/u02/oracle10g/oradata/rjbdb/undotbs01.dbf'
   SIZE 200M REUSE AUTOEXTEND ON MAXSIZE UNLIMITED;

9. A couple of things are worth noting in this example. We are explicitly setting the passwords for SYS and SYSTEM; if we didn’t specify them here, they will default to change_on_install and manager, respectively.
10. The redo log file groups have only one member each; once our database is in production, we should multiplex them. Because we specified an undo tablespace with the UNDO_TABLESPACE parameter in the initialization parameter file, we need to create that tablespace here; otherwise, the instance will not start.

11. After the `create database` command completes successfully, the database is mounted and opened for use.

12. Create additional tablespaces for users, indexes, and applications.

13. Build data dictionary views with the supplied scripts `catalog.sql` and `catproc.sql`. The script `catalog.sql` creates views against the data dictionary tables, dynamic performance views, and public synonyms for most of the views. The group PUBLIC is granted read-only access to the views. The script `catproc.sql` sets up PL/SQL.

14. Back up the database using either a cold backup or the Recovery Manager. In case of a database failure in the early stages of deployment, you have a complete and running database to fall back on, and most likely you will not have to re-create the database from scratch.