Basic SQL SELECT Statements

INTRODUCTION TO ORACLE9i: SQL EXAM

OBJECTIVES COVERED IN THIS CHAPTER:

✓ Writing Basic SQL Select Statements
  ▪ List the capabilities of SQL SELECT statements
  ▪ Execute a basic SELECT statement

✓ Restricting and Sorting Data
  ▪ Limit the rows retrieved by a query
  ▪ Sort the rows retrieved by a query

Exam objectives are subject to change at any time without prior notice and at Oracle’s sole discretion. Please visit Oracle’s Certification website (http://www.oracle.com/education/certification/) for the most current exam objectives listing.
The Oracle9i database provides many useful and powerful features. Many of the features are incorporated at the SQL level. SQL (pronounced “sequel”) has been adopted by most relational database management systems (RDBMS). The American National Standards Institute (ANSI) has been refining standards for the SQL language for the past 20 years. Oracle, like many other companies, has taken the ANSI standard of SQL and extended it to include much additional functionality.

SQL is the basic language used to manipulate and retrieve data from the Oracle9i database. SQL is a nonprocedural language—it does not have programmatic constructs such as loop structures. PL/SQL is Oracle’s procedural extension of SQL, and SQLJ allows embedded SQL operations in Java code. The scope of this test includes only SQL.

The SQL SELECT statement is used to query data from the database-storage structures, such as tables and views. In this chapter, you will learn how to write basic SQL statements to retrieve data from tables. You will also learn how to limit the information retrieved and to display the results in a specific order.

SQL Fundamentals

The basic structure of data storage in Oracle9i database is a table. A table consists of columns and its characteristics. Data is stored in the table as rows. Creating and maintaining tables are discussed in detail in Chapter 7, “Managing Tables and Constraints.” To get started with SQL in this chapter, you will use the sample HR schema supplied with the Oracle9i database.
When you install Oracle software, choose the option to create a seed database. This database will have the sample schemas used in this book. The default IDs and password for the seed database are SYSTEM/_MANAGER, SYS/CHANGE_ON_INSTALL. The account SYS is the Oracle dictionary owner, and SYSTEM is a DBA account. Initially, the sample schemas are locked. You need to connect to the database using SYSTEM, and then unlock the account using the ALTER USER statement. To unlock the HR schema, use ALTER USER HR IDENTIFIED BY HRPASSWORD ACCOUNT UNLOCK; Now you can connect to the HR schema using the password HRPASSWORD.

SQL statements are like plain English but with specific syntax. SQL is a simple, yet powerful, language used to create, access, and manipulate data and structure in the database. SQL statements can be categorized as listed in Table 1.1.

<table>
<thead>
<tr>
<th>SQL Category</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Manipulation Language (DML)</td>
<td>Used to access, create, modify, or delete data in the existing structures of the database. DML statements include those to query information (SELECT), add new rows (INSERT), modify existing rows (UPDATE), delete existing rows (DELETE), perform a conditional update or insert operation (MERGE), see an execution plan of SQL (EXPLAIN PLAN), and lock a table to restrict access (LOCK TABLE).</td>
</tr>
<tr>
<td>Data Definition Language (DDL)</td>
<td>Used to define, alter, or drop database objects and their privileges. DDL statements include those to create, modify, drop, or rename objects (CREATE, ALTER, DROP, RENAME), remove all rows from a database object without dropping the structure (TRUNCATE), manage access privileges (GRANT, REVOKE), audit database use (AUDIT, NOAUDIT) and add a description about an object to the dictionary (COMMENT).</td>
</tr>
</tbody>
</table>
Chapter 1 • Basic SQL SELECT Statements

Table 1.1 provides an overview of all the statements that will be covered in this book. Do not worry if you do not understand certain terms, such as role, session, privilege, and so on. We will cover all the statements in the coming chapters with many examples. In this chapter, we will begin with writing simple statements to query the database (SELECT statements). But first, we need to review some SQL fundamentals.

Oracle Datatypes

When you create a table to store data in the database, you need to specify a datatype for all of the columns you define in the table. Oracle has many datatypes to suit application requirements. Oracle9i also supports ANSI and DB2 datatypes. The Oracle built-in datatypes can be broadly classified as shown in Table 1.2.
In this section, we will discuss only a few of the built-in datatypes to get started with SQL. All the datatypes and their usage are discussed in detail in Chapter 7.

**CHAR**(<size>**)

The *CHAR* datatype is a fixed-length alphanumeric string, which has a maximum length in bytes. Data stored in CHAR columns is space-padded to fill the maximum length. Its size can range from a minimum of 1 byte to a maximum of 2000 bytes. The default size is 1.

When you create a column using the CHAR datatype, the database will ensure that all data placed in this column has the defined length. If the data is shorter than the defined length, it is space-padded on the right to the specified length. If the data is longer, an error is raised.

**VARCHAR2**(<size>**)

The *VARCHAR2* datatype is a variable-length alphanumeric string, which has a maximum length in bytes. VARCHAR2 columns require only the amount of space needed to store the data and can store up to 4000 bytes. There is no default size for the VARCHAR2 datatype. An empty VARCHAR2(2000) column takes up as much room in the database as an empty VARCHAR2(2) column.
The default size of a CHAR datatype is 1. For a VARCHAR2 datatype, you must always specify the size.

The VARCHAR2 and CHAR datatypes have different comparison rules for trailing spaces. With the CHAR datatype, trailing spaces are ignored. With the VARCHAR2 datatype, trailing spaces are not ignored, and they sort higher than no trailing spaces. Here’s an example:

- CHAR datatype: ‘Yo’ = ‘Yo   ’
- VARCHAR2 datatype: ‘Yo’ < ‘Yo   ’

**NUMBER (<p>, <s>)**

The NUMBER datatype stores numbers with a *precision* of *p* digits and a *scale* of *s* digits. The precision and scale values are optional. Numeric datatypes are used to store negative and positive integers, fixed-point numbers, and floating-point numbers. The precision can be between 1 and 38, and the scale has a range between –84 and 127. If the precision and scale are omitted, Oracle assumes the maximum of the range for both values.

You can have precision and scale digits in the integer part. The scale rounds the value after the decimal point to *s* digits. For example, if you define a column as NUMBER(5,2), the range of values you can store in this column is from –999.99 to 999.99; that is, 5–2=3 for the integer part, and the decimal part is rounded to two digits. Even if you do not include the decimal part for the value inserted, the maximum number you can store in a NUMBER(5,2) definition is 999.

Oracle will round numbers inserted into numeric columns with a scale smaller than the inserted number. For example, if a column were defined as NUMBER(4,2) and you specified a value of 12.125 to go into that column, the resulting number would be rounded to 12.13 before it was inserted into the column. If the value exceeds the precision, however, an Oracle error is returned. You cannot insert 123.1 into a column defined as NUMBER(4,2). Specifying the scale and precision does not force all inserted values to be a fixed length.

If the scale is negative, the number is rounded to the left of the decimal. Basically, a negative scale forces *s* number of zeros just to the left of the decimal.
If you specify a scale that is greater than the precision value, the precision defines the maximum number of digits to right of the decimal point after the zeros. For example, if a column is defined as `NUMBER(3,5)`, the range of values you can store is from $-0.00999$ to $0.00999$; that is, it requires two zeros ($s-p$) after the decimal point and rounds the decimal part to three digits ($p$) after zeros. Table 1.3 shows several examples of how numeric data is stored with various definitions.

**Table 1.3** Precision and Scale Examples

<table>
<thead>
<tr>
<th>Value</th>
<th>Datatype</th>
<th>Stored Value</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>123.2564</td>
<td>NUMBER</td>
<td>123.2564</td>
<td>Range and precision are set to the maximum, so the datatype can store any value.</td>
</tr>
<tr>
<td>1234.9876</td>
<td>NUMBER(6,2)</td>
<td>1234.99</td>
<td>Since scale is only 2, the decimal part of the value is rounded to two digits.</td>
</tr>
<tr>
<td>12345.12345</td>
<td>NUMBER(6,2)</td>
<td>Error</td>
<td>The range of integer part is only from $-9999$ to $9999$.</td>
</tr>
<tr>
<td>123456</td>
<td>NUMBER(6,2)</td>
<td>Error</td>
<td>Precision is larger than specified; range is only from $-9999$ to $9999$.</td>
</tr>
<tr>
<td>1234.9876</td>
<td>NUMBER(6)</td>
<td>1235</td>
<td>Decimal part rounded to the next integer.</td>
</tr>
<tr>
<td>123456.1</td>
<td>NUMBER(6)</td>
<td>123456</td>
<td>Decimal part rounded.</td>
</tr>
<tr>
<td>12345.345</td>
<td>NUMBER(5,-2)</td>
<td>12300</td>
<td>Negative scale rounds the number &lt;$s$&gt; digits left to the decimal point. –2 rounds to hundreds.</td>
</tr>
<tr>
<td>1234567</td>
<td>NUMBER(5,-2)</td>
<td>1234600</td>
<td>Rounded to the nearest hundred.</td>
</tr>
</tbody>
</table>
Chapter 1 • Basic SQL SELECT Statements

**DATE**

The *DATE* datatype is used to store date and time information. This datatype can be converted to other forms for viewing, but it has a number of special functions and properties that make date manipulation and calculations simple. The time component of the *DATE* datatype has a resolution of one second—

---

**TABLE 1.3** Precision and Scale Examples *(continued)*

<table>
<thead>
<tr>
<th>Value</th>
<th>Datatype</th>
<th>Stored Value</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>12345678</td>
<td>NUMBER(5,-2)</td>
<td>Error</td>
<td>Outside range; can have only five digits, excluding the two zeros representing hundreds, for a total of seven digits. ( s-p = s+p = 5+2 = 7 )</td>
</tr>
<tr>
<td>123456789</td>
<td>NUMBER(5,-4)</td>
<td>123460000</td>
<td>Rounded to nearest 10000.</td>
</tr>
<tr>
<td>1234567890</td>
<td>NUMBER(5,-4)</td>
<td>Error</td>
<td>Outside range; can have only five digits excluding the four trailing zeros.</td>
</tr>
<tr>
<td>12345.58</td>
<td>NUMBER(*, 1)</td>
<td>12345.6</td>
<td>Use of * in precision specifies the default limit (38).</td>
</tr>
<tr>
<td>0.1</td>
<td>NUMBER(4,5)</td>
<td>Error</td>
<td>Requires a zero after the decimal point (5–4=1)</td>
</tr>
<tr>
<td>0.01234567</td>
<td>NUMBER(4,5)</td>
<td>0.01235</td>
<td>Rounded to four digits after the decimal point and zero.</td>
</tr>
<tr>
<td>0.09999</td>
<td>NUMBER(4,5)</td>
<td>0.09999</td>
<td>Stored as it is; only four digits after the decimal point and zero.</td>
</tr>
<tr>
<td>0.099996</td>
<td>NUMBER(4,5)</td>
<td>Error</td>
<td>Rounding this value to four digits after the decimal and zero results in 0.1, which is outside the range.</td>
</tr>
</tbody>
</table>
no less. The DATE datatype occupies a storage space of seven bytes. The
following information is contained within each DATE datatype:

- Century
- Year
- Month
- Day
- Hour
- Minute
- Second

Date values are inserted or updated in the database by converting either a
numeric or character value into a DATE datatype using the function TO_DATE.
Oracle defaults the format to display date as DD-MON-YY. This format
shows that the default date must begin with a two-digit day, followed by a
three-character abbreviation for the month, followed by a two-digit year. If
you specify the date without including a time component, the time is defaulted
to midnight, or 00:00:00 in military time. The SYSDATE function returns
the current system date and time from the database server to which you’re
currently connected.

The default date format is specified using the initialization parameter NLS_ DATE_FORMAT. The value of this parameter can be changed in the user’s
environment or in the user’s session.

Operators and Literals

An operator is a manipulator that is applied to a data item in order to return a
result. Special characters represent different operations in Oracle (+ represents
addition, for example). Operators are commonly used in all programming
environments, and you should already be familiar with the following operators,
which may be classified into two types:

**Unary operator**  A unary operator has only one operand. Examples are
+2 and −5. They have the format `<operator><operand>`.
Binary operator  A binary operator has two operands. Examples are 5+4 and 7*5. They have the format <operand1><operator><operand2>. You can insert spaces between the operand and operator to improve readability.

**Arithmetic Operators**

Arithmetic operators operate on numeric values. Table 1.4 shows the various arithmetic operators in Oracle and how to use them.

**TABLE 1.4** Arithmetic Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Purpose</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ -</td>
<td>Unary operators: Use to represent positive or negative data item. For positive items, the + is optional.</td>
<td>-234.44</td>
</tr>
<tr>
<td>+</td>
<td>Addition: Use to add two data items or expressions.</td>
<td>2+4</td>
</tr>
<tr>
<td>-</td>
<td>Subtraction: Use to find the difference between two data items or expressions.</td>
<td>20.4-2</td>
</tr>
<tr>
<td>*</td>
<td>Multiplication: Use to multiply two data items or expressions.</td>
<td>5*10</td>
</tr>
<tr>
<td>/</td>
<td>Division: Use to divide a data item or expression with another.</td>
<td>8.4/2</td>
</tr>
</tbody>
</table>

**NOTE**

Do not use two hyphens (--) to represent double negation; use a space or parenthesis in between, as in -(20). Two hyphens represent the beginning of a comment in SQL.

**Concatenation Operator**

The concatenation operator is used to concatenate or join two character (text) strings. The result of concatenation is another character string. Concatenating a zero-length string '' or a NULL with another string results in a string, not a NULL. Two vertical bars || are used as the concatenation operator.
Here are two examples:

- 'Oracle9i' || 'Database' results in 'Oracle9iDatabase'
- 'Oracle9i' || 'Database' results in 'Oracle9i Database'

**Set Operators**

*Set operators* are used in compound queries—queries that combine the results of two queries. The number of columns selected in both queries must be the same. Table 1.5 lists the set operators and how to use them. Set operators are discussed in detail in Chapter 5, “Joins and Subqueries.”

**TABLE 1.5** Set Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNION</td>
<td>Returns all rows from either queries; no duplicate rows</td>
</tr>
<tr>
<td>UNION ALL</td>
<td>Returns all rows from either query, including duplicates</td>
</tr>
<tr>
<td>INTERSECT</td>
<td>Returns distinct rows that are returned by both queries</td>
</tr>
<tr>
<td>MINUS</td>
<td>Returns distinct rows that are returned by the first query but not returned by the second.</td>
</tr>
</tbody>
</table>

**Operator Precedence**

If multiple operators are used in the same expression, Oracle evaluates them in the *order of precedence* set in the database engine. Operators with higher precedence are evaluated before operators with lower precedence. Operators with the same precedence are evaluated from left to right. Table 1.6 lists the precedence.

**TABLE 1.6** SQL Operator Precedence

<table>
<thead>
<tr>
<th>Precedence</th>
<th>Operator</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>- +</td>
<td>Unary operators, negation</td>
</tr>
<tr>
<td>2</td>
<td>* /</td>
<td>Multiplication, division</td>
</tr>
<tr>
<td>3</td>
<td>+ -</td>
<td></td>
</tr>
</tbody>
</table>
Using parentheses changes the order of precedence. The innermost parenthesis is evaluated first. In the expression 1+2*3, the result is 7, because 2*3 is evaluated first and the result is added to 1. In the expression (1+2)*3, 1+2 is evaluated first, and the result is multiplied by 3, giving 9.

**Literals**

*Literals* are values that represent a fixed value (constant). There are four types of literals:

- Text (or character)
- Integer
- Number
- Interval

You can use literals within many of the SQL functions, expressions, and conditions.

**Text**

The text literal must be enclosed in single quotation marks. Any character between the quotation marks is considered part of the text value. Oracle treats all text literals as though they were CHAR datatypes for comparison (blank padded). The maximum length of a text literal is 4000 bytes. Single quotation marks can be included in the literal text value by preceding it with another single quotation mark. Here are some examples of text literals:

- 'The Quick Brown Fox'
- 'That man''s suit is black'
- 'And I quote: “This will never do.” '
- '12-SEP-2001'

**Integer**

Integer literals can be any number of numerals, excluding a decimal separator and up to 38 digits long. Here are two examples:

- 24
- −456
Number
Number literals can include scientific notation, as well as digits and the decimal separator. Here are some examples:

- 24
- -345.65
- 23E-10

Interval
Interval literals specify a period of time in terms of years and months or in terms of days and seconds. These literals correspond to the Oracle datatype INTERVAL YEAR TO MONTH and INTERVAL DAY TO SECOND. These datatypes will be discussed in more detail in Chapter 7.

Writing Simple Queries

A query is a request for information from the database tables. Simple queries are those that retrieve data from a single table. The basis of a query is the SELECT statement. Queries using multiple tables are discussed in later chapters.

Using the SELECT Statement

The SELECT statement is the most commonly used statement in SQL. It allows you to retrieve information already stored in the database. The statement begins with the keyword SELECT, followed by the column names whose data you want to query. You can either select information from all the columns (denoted by *) or name specific columns in the SELECT clause to retrieve data. The FROM clause provides the name of the table, view, or materialized view to use in the query. These objects are discussed in detail in later chapters. For simplicity, we will use tables for the rest of this chapter.

Let’s use the JOBS table defined in the HR schema of the Oracle9i seed database created during installation. The JOBS table definition is provided in Table 1.7.
Chapter 1 • Basic SQL SELECT Statements

**Table 1.7** JOBS Table Definition

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Datatype</th>
<th>length</th>
</tr>
</thead>
<tbody>
<tr>
<td>JOB_ID</td>
<td>VARCHAR2</td>
<td>10</td>
</tr>
<tr>
<td>JOB_TITLE</td>
<td>VARCHAR2</td>
<td>30</td>
</tr>
<tr>
<td>MIN_SALARY</td>
<td>NUMBER</td>
<td>6,0</td>
</tr>
<tr>
<td>MAX_SALARY</td>
<td>NUMBER</td>
<td>6,0</td>
</tr>
</tbody>
</table>

The simple form of a SELECT statement to retrieve all the columns and rows from the JOBS table is as follows (only part of output result set is shown here):

```
SQL> SELECT * FROM jobs;

JOB_ID     JOB_TITLE                       MIN_SALARY  MAX_SALARY
---------- ------------------------------- ---------- ----------
AD_PRES    President                            20000      40000
AD_VP      Administration Vice President        15000      30000
AD_ASST    Administration Assistant              3000       6000
FI_MGR     Finance Manager                       8200      16000
FI_ACCOUNT Accountant                            4200       9000
...         ...                               ...        ...
IT_PROG    Programmer                            4000        10000
MK_MAN     Marketing Manager                     9000      15000
MK_REP     Marketing Representative              4000        9000
HR_REP     Human Resources Representative        4000        9000
PR_REP     Public Relations Representative       4500      10500

19 rows selected.
SQL>
```

The keywords, column names, and table names are case insensitive. Only literals enclosed in single quotation marks are case sensitive in Oracle.
How do you list only the job title and minimum salary from this table? If you know the column names and the table name, writing the query is simple. Here, the column names are JOB_TITLE and MIN_SALARY, and the table name is JOBS. Execute the query by ending the query with a semicolon. In SQL*Plus, you can execute the query by entering a slash on a line by itself or by using the RUN command.

```
SQL> SELECT job_title, min_salary FROM jobs;

                JOB_TITLE              MIN_SALARY
------------------------------- ----------
              President              20000
   Administration Vice President   15000
   Administration Assistant        3000
              Finance Manager        8200
               Accountant            4200
   Accounting Manager              8200
               Public Accountant      4200
                             ... ...
               Programmer             4000
         Marketing Manager          9000
    Marketing Representative        4000
   Human Resources Representative  4000
   Public Relations Representative 4500

19 rows selected.
SQL>
```

Notice that the numeric column (MIN_SALARY) is aligned to the right and the character column (JOB_TITLE) is aligned to the left. Does it seem that the column heading MIN_SALARY should be more meaningful? Well, you can provide a column alias to appear in the query results.

**Column Alias Names**

The column alias name is defined next to the column name with a space or by using the keyword AS. If you want a space in the column alias name, you must enclose it in double quotation marks. The case is preserved only when the alias name is enclosed in double quotation marks; otherwise, the display
will be uppercase. The following example demonstrates using an alias name for the column heading in the previous query.

```sql
SQL> SELECT job_title AS Title,
          min_salary AS 'Minimum Salary' FROM jobs
SQL> /

<table>
<thead>
<tr>
<th>TITLE</th>
<th>Minimum Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>President</td>
<td>20000</td>
</tr>
<tr>
<td>Administration Vice President</td>
<td>15000</td>
</tr>
<tr>
<td>Administration Assistant</td>
<td>3000</td>
</tr>
<tr>
<td>Finance Manager</td>
<td>8200</td>
</tr>
<tr>
<td>Accountant</td>
<td>4200</td>
</tr>
<tr>
<td>Accounting Manager</td>
<td>8200</td>
</tr>
<tr>
<td>... ... ...</td>
<td></td>
</tr>
<tr>
<td>Programmer</td>
<td>4000</td>
</tr>
<tr>
<td>Marketing Manager</td>
<td>9000</td>
</tr>
<tr>
<td>Marketing Representative</td>
<td>4000</td>
</tr>
<tr>
<td>Human Resources Representative</td>
<td>4000</td>
</tr>
<tr>
<td>Public Relations Representative</td>
<td>4500</td>
</tr>
</tbody>
</table>

19 rows selected.
SQL>
```

In this listing, the column alias name Title appears in all capital letters because we did not enclose it in double quotation marks.

The asterisk (*) is used to select all columns in the table. This is very useful when you do not know the column names or when you are too lazy to type all of the column names.

**Ensuring Uniqueness**

The `DISTINCT` keyword (or `UNIQUE` keyword) following `SELECT` ensures that the resulting rows are unique. Uniqueness is verified against the complete row,
not the first column. If you need to find the unique departments in the EMPLOYEES table, issue this query:

    SQL> SELECT DISTINCT department_id FROM employees;

    DEPARTMENT_ID
    -------------
    10
    20
    30
    40
    50
    60
    70
    80
    90
    100
    110

    12 rows selected.
    SQL>

To demonstrate that uniqueness is enforced across the row, let's do one more query using the SELECT DISTINCT clause. Notice DEPARTMENT_ID repeating for each JOB_ID value.

    SQL> SELECT DISTINCT department_id, job_id FROM employees;

    DEPARTMENT_ID   JOB_ID
    -------------   ----------
    10   AD_ASST
    20   MK_MAN
    20   MK_REP
    30   PU_CLERK
    30   PU_MAN
    40   HR_REP
    50   SH_CLERK
    50   ST_CLERK
    50   ST_MAN
Chapter 1 • Basic SQL SELECT Statements

SELECT * FROM TAB; shows all the tables and views in your schema.

The DUAL Table

The DUAL table is a dummy table available to all users in the database. It has one column and one row. The DUAL table is used to select system variables or to evaluate an expression. Here are a few examples:

SQL> SELECT SYSDATE, USER FROM dual;

<table>
<thead>
<tr>
<th>SYSDATE</th>
<th>USER</th>
</tr>
</thead>
<tbody>
<tr>
<td>18-SEP-02</td>
<td>HR</td>
</tr>
</tbody>
</table>

SQL> SELECT 'I''m ' || user || ' Today is ' || SYSDATE
2  FROM dual;

'I''M'||USER||'TODAY IS'||SYSDATE
-----------------------------------------------------
I'm HR Today is 18-SEP-02

SYSDATE and USER are built-in functions that provide information about the environment. These functions are discussed in Chapter 3, “Single-Row Functions.”
Limiting Rows

A WHERE clause in the SELECT statement is used to limit the number of rows processed. Any logical conditions of the WHERE clause use the comparison operators. Rows are returned or operated upon where the data satisfies the logical condition(s) of the WHERE clause. You can use column names or expressions in the WHERE clause, but not column alias names. The WHERE clause follows the FROM clause in the SELECT statement.

How do you list the employees who work for department 90? The following example shows how to limit the query to only the records belonging to department 90 by using a WHERE clause.

```
SQL>SELECT first_name ||''||last_name 'Name',
2       department_id
3  FROM employees
4  WHERE department_id =90;
```

<table>
<thead>
<tr>
<th>Name</th>
<th>DEPARTMENT_ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Steven King</td>
<td>90</td>
</tr>
<tr>
<td>Neena Kochhar</td>
<td>90</td>
</tr>
<tr>
<td>Lex De Haan</td>
<td>90</td>
</tr>
</tbody>
</table>

You need not include the column names in the SELECT clause to use them in the WHERE clause.

Various operators available in Oracle9i can be used in the WHERE clause to limit the number of rows.

Comparison Operators

Comparison operators compare two values or expressions and give a Boolean result of TRUE, FALSE, or NULL. The comparison operators include those that test for equality, inequality, less than, greater than, and value comparisons.

= (Equality)
The = operator tests for equality. The test evaluates to TRUE if the values or results of an expression on both sides of the operator are equal.
Chapter 1 • Basic SQL SELECT Statements

SQL> SELECT first_name ||''||last_name "Name",
2       department_id
3  FROM employees
4  WHERE department_id =90;

<table>
<thead>
<tr>
<th>Name</th>
<th>DEPARTMENT_ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Steven King</td>
<td>90</td>
</tr>
<tr>
<td>Neena Kochhar</td>
<td>90</td>
</tr>
<tr>
<td>Lex De Haan</td>
<td>90</td>
</tr>
</tbody>
</table>

!=, <>, or ^= (Inequality)
You can use any one of these three operators to test for inequality. The test evaluates to TRUE if the values on both sides of the operator do not match. The operator <> works on all platforms, the use of other operators for inequality checking is not supported in all platforms.

SQL> SELECT first_name ||''||last_name "Name",
2       commission_pct
3  FROM employees
4  WHERE commission_pct !=.35;

<table>
<thead>
<tr>
<th>Name</th>
<th>COMMISSION_PCT</th>
</tr>
</thead>
<tbody>
<tr>
<td>John Russell</td>
<td>.4</td>
</tr>
<tr>
<td>Karen Partners</td>
<td>.3</td>
</tr>
<tr>
<td>Alberto Errazuriz</td>
<td>.3</td>
</tr>
<tr>
<td>Gerald Cambrault</td>
<td>.3</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>Jack Livingston</td>
<td>.2</td>
</tr>
<tr>
<td>Kimberely Grant</td>
<td>.15</td>
</tr>
<tr>
<td>Charles Johnson</td>
<td>.1</td>
</tr>
</tbody>
</table>

32 rows selected.
SQL>
< (Less Than)
The < operator evaluates to TRUE if the left side (expression or value) of the operator is less than the right side of the operator.

```
SQL> SELECT first_name ||''||last_name 'Name',
           commission_pct
FROM employees
WHERE commission_pct <.15;
```

<table>
<thead>
<tr>
<th>Name</th>
<th>COMMISSION_PCT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mattea Marvins</td>
<td>.1</td>
</tr>
<tr>
<td>David Lee</td>
<td>.1</td>
</tr>
<tr>
<td>Sundar Ande</td>
<td>.1</td>
</tr>
<tr>
<td>Amit Banda</td>
<td>.1</td>
</tr>
<tr>
<td>Sundita Kumar</td>
<td>.1</td>
</tr>
<tr>
<td>Charles Johnson</td>
<td>.1</td>
</tr>
</tbody>
</table>

6 rows selected.
SQL>

> (More Than)
The > operator evaluates to TRUE if the left side (expression or value) of the operator is greater than the right side of the operator.

```
SQL> SELECT first_name ||''||last_name 'Name',
           commission_pct
FROM employees
WHERE commission_pct >.35;
```

<table>
<thead>
<tr>
<th>Name</th>
<th>COMMISSION_PCT</th>
</tr>
</thead>
<tbody>
<tr>
<td>John Russell</td>
<td>.4</td>
</tr>
</tbody>
</table>

SQL>

<= (Less Than or Equal to)
The <= operator evaluates to TRUE if the left side (expression or value) of the operator is less than or equal to the right side of the operator.
Chapter 1 • Basic SQL SELECT Statements

SQL> SELECT first_name ||''||last_name "Name",
                commission_pct
  FROM employees
WHERE commission_pct <=.15;

<table>
<thead>
<tr>
<th>Name</th>
<th>COMMISSION_PCT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Oliver Tuvault</td>
<td>.15</td>
</tr>
<tr>
<td>Danielle Greene</td>
<td>.15</td>
</tr>
<tr>
<td>Mattea Marvins</td>
<td>.1</td>
</tr>
<tr>
<td>David Lee</td>
<td>.1</td>
</tr>
<tr>
<td>Sundar Ande</td>
<td>.1</td>
</tr>
<tr>
<td>Amit Banda</td>
<td>.1</td>
</tr>
<tr>
<td>William Smith</td>
<td>.15</td>
</tr>
<tr>
<td>Elizabeth Bates</td>
<td>.15</td>
</tr>
<tr>
<td>Sundita Kumar</td>
<td>.1</td>
</tr>
<tr>
<td>Kimberely Grant</td>
<td>.15</td>
</tr>
<tr>
<td>Charles Johnson</td>
<td>.1</td>
</tr>
</tbody>
</table>

11 rows selected.
SQL>

>= (Greater Than or Equal to)

The >= operator evaluates to TRUE if the left side (expression or value) of the operator is greater than or equal to the right side of the operator.

SQL> SELECT first_name ||''||last_name "Name",
                commission_pct
  FROM employees
WHERE commission_pct >=.35;

<table>
<thead>
<tr>
<th>Name</th>
<th>COMMISSION_PCT</th>
</tr>
</thead>
<tbody>
<tr>
<td>John Russell</td>
<td>.4</td>
</tr>
<tr>
<td>Janette King</td>
<td>.35</td>
</tr>
<tr>
<td>Patrick Sully</td>
<td>.35</td>
</tr>
<tr>
<td>Allan McEwen</td>
<td>.35</td>
</tr>
</tbody>
</table>

SQL>
ANY or SOME

The ANY or SOME operators are used to compare a value to each value in a list or subquery. The ANY and SOME operators always must be precede by the comparison operators =, !=, <, <=, or >.

```
SQL>SELECT first_name ||''||last_name 'Name',
       2       department_id
       3  FROM employees
       4  WHERE department_id <=ANY (10,15,20,25);
```

<table>
<thead>
<tr>
<th>Name</th>
<th>DEPARTMENT_ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jennifer Whalen</td>
<td>10</td>
</tr>
<tr>
<td>Michael Hartstein</td>
<td>20</td>
</tr>
<tr>
<td>Pat Fay</td>
<td>20</td>
</tr>
</tbody>
</table>

```
SQL>
```

ALL

The ALL operator is used to compare a value to every value in a list or subquery. The ALL operator must always be preceded by the comparison operators =, !=, <, <=, or >.

```
SQL>SELECT first_name ||''||last_name 'Name',
       2       department_id
       3  FROM employees
       4  WHERE department_id >=ALL (80,90,100);
```

<table>
<thead>
<tr>
<th>Name</th>
<th>DEPARTMENT_ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nancy Greenberg</td>
<td>100</td>
</tr>
<tr>
<td>Daniel Faviet</td>
<td>100</td>
</tr>
<tr>
<td>John Chen</td>
<td>100</td>
</tr>
<tr>
<td>Ismael Sciarra</td>
<td>100</td>
</tr>
<tr>
<td>Jose Manuel Urman</td>
<td>100</td>
</tr>
<tr>
<td>Luis Popp</td>
<td>100</td>
</tr>
<tr>
<td>Shelley Higgins</td>
<td>110</td>
</tr>
<tr>
<td>William Gietz</td>
<td>110</td>
</tr>
</tbody>
</table>

8 rows selected.

SQL>
For all the comparison operators discussed, if one side of the operator is NULL, the result is NULL.

**Logical Operators**

*Logical operators* are used to combine the results of two comparison conditions to produce a single result or to reverse the result of a single comparison. **NOT**, **AND**, and **OR** are the logical operators.

**NOT**

The **NOT** operator is used to reverse the result. It evaluates to TRUE if the operand is FALSE, evaluates to FALSE if the operand is TRUE. **NOT** returns NULL if the operand is NULL.

```
SQL> SELECT first_name, department_id
2    FROM employees
3* WHERE not (department_id >= 30);
```

<table>
<thead>
<tr>
<th>FIRST_NAME</th>
<th>DEPARTMENT_ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jennifer</td>
<td>10</td>
</tr>
<tr>
<td>Michael</td>
<td>20</td>
</tr>
<tr>
<td>Pat</td>
<td>20</td>
</tr>
</tbody>
</table>

**AND**

The **AND** operator evaluates to TRUE if both operands are TRUE. It evaluates to FALSE if either operand is FALSE. Otherwise, it returns NULL.

```
SQL> SELECT first_name, salary
2    FROM employees
3* WHERE last_name = 'Smith'
4* AND salary > 7500;
```

<table>
<thead>
<tr>
<th>FIRST_NAME</th>
<th>SALARY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lindsey</td>
<td>8000</td>
</tr>
</tbody>
</table>

**OR**

The **OR** operator evaluates to TRUE if either operand is TRUE. It evaluates to FALSE if both operands are FALSE. Otherwise, it returns NULL.
Writing Simple Queries

SQL> SELECT first_name, last_name
2  FROM   employees
3  WHERE  first_name = 'Kelly'
4* OR     last_name  = 'Smith';

<table>
<thead>
<tr>
<th>FIRST_NAME</th>
<th>LAST_NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lindsey</td>
<td>Smith</td>
</tr>
<tr>
<td>William</td>
<td>Smith</td>
</tr>
<tr>
<td>Kelly</td>
<td>Chung</td>
</tr>
</tbody>
</table>

Logical Operator Truth Tables

The following tables can be used as truth tables for the three logical operators.

AND Truth Table

<table>
<thead>
<tr>
<th>AND</th>
<th>TRUE</th>
<th>FALSE</th>
<th>NULL</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>TRUE</td>
<td>FALSE</td>
<td>NULL</td>
</tr>
<tr>
<td>FALSE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>FALSE</td>
</tr>
<tr>
<td>NULL</td>
<td>NULL</td>
<td>FALSE</td>
<td>NULL</td>
</tr>
</tbody>
</table>

OR Truth Table

<table>
<thead>
<tr>
<th>OR</th>
<th>TRUE</th>
<th>FALSE</th>
<th>NULL</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
</tr>
<tr>
<td>FALSE</td>
<td>TRUE</td>
<td>FALSE</td>
<td>NULL</td>
</tr>
<tr>
<td>NULL</td>
<td>TRUE</td>
<td>NULL</td>
<td>NULL</td>
</tr>
</tbody>
</table>

NOT Truth Table

<table>
<thead>
<tr>
<th>NOT</th>
<th>TRUE</th>
<th>FALSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>FALSE</td>
<td></td>
</tr>
<tr>
<td>FALSE</td>
<td>TRUE</td>
<td></td>
</tr>
<tr>
<td>NULL</td>
<td>NULL</td>
<td></td>
</tr>
</tbody>
</table>
Other Operators

In this section, we will discuss all the operators that can be used in the WHERE clause of the SQL statement that were not discussed earlier.

**IN and NOT IN**

The IN and NOT IN operators are used to test a membership condition. IN is equivalent to the =ANY operator, which evaluates to TRUE if the value exists in the list or the result set from a subquery. The NOT IN operator is equivalent to the !=ALL operator, which evaluates to TRUE if the value does not exist in the list or the result set from a subquery. The following examples demonstrate the use of these two operators.

```
SQL> SELECT first_name, last_name, department_id
2  FROM   employees
3  WHERE  department_id IN (10, 20, 90);

FIRST_NAME           LAST_NAME              DEPARTMENT_ID
-------------------- ------------------------- ----------
Steven               King                              90
Neena                Kochhar                           90
Lex                  De Haan                           90
Jennifer             Whalen                            10
Michael              Hartstein                         20
Pat                  Fay                               20

6 rows selected.

SQL> SELECT first_name, last_name, department_id
2  FROM   employees
3  WHERE  department_id NOT IN
4*        (10, 30, 40, 50, 60, 80, 90, 110, 100)

SQL> /
```
### Other Operators

<table>
<thead>
<tr>
<th>FIRST_NAME</th>
<th>LAST_NAME</th>
<th>DEPARTMENT_ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Michael</td>
<td>Hartstein</td>
<td>20</td>
</tr>
<tr>
<td>Pat</td>
<td>Fay</td>
<td>20</td>
</tr>
<tr>
<td>Hermann</td>
<td>Baer</td>
<td>70</td>
</tr>
</tbody>
</table>

#### NOT IN

When using the NOT IN operator, if any value in the list or the result returned from the subquery is NULL, the query returns no rows. For example, `last_name not in ('Smith', 'Thomas', NULL)` evaluates to `last_name != 'Smith' AND last_name != 'Thomas' AND last_name != NULL`. Any comparison on a NULL value results in NULL.

#### BETWEEN

The BETWEEN operator is used to test a range. BETWEEN A AND B evaluates to TRUE if the value is greater than or equal to A and less than or equal to B. If NOT is used, the result is the reverse. The following example lists all the employees whose salary is between $5,000 and $6,000.

```sql
SQL> SELECT first_name, last_name, salary
  2 FROM employees
  3* WHERE salary BETWEEN 5000 AND 6000;
```

<table>
<thead>
<tr>
<th>FIRST_NAME</th>
<th>LAST_NAME</th>
<th>SALARY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bruce</td>
<td>Ernst</td>
<td>6000</td>
</tr>
<tr>
<td>Kevin</td>
<td>Mourgos</td>
<td>5800</td>
</tr>
<tr>
<td>Pat</td>
<td>Fay</td>
<td>6000</td>
</tr>
</tbody>
</table>

#### EXISTS

The EXISTS operator is always followed by a subquery in parentheses. (For more information on subqueries, refer to Chapter 5.) EXISTS evaluates to TRUE if the subquery returns at least one row. The following example lists the employees who work for Administration department.

```sql
SQL> SELECT first_name, last_name, department_id
  2 FROM employees
  3* WHERE department_id = 20;
```
SQL> SELECT last_name, first_name, department_id
2  FROM   employees e
3  WHERE  EXISTS (select 1 FROM departments d
4     WHERE  d.department_id = e.department_id
5*     AND    d.department_name = 'Administration');

LAST_NAME              FIRST_NAME           DEPARTMENT_ID
---------------------- -------------------- -------------
Whalen                 Jennifer                        10

**IS NULL and IS NOT NULL**

To find the NULL values or NOT NULL values, you need to use the IS NULL operator. The = or != operator will not work with NULL values. IS NULL evaluates to TRUE if the value is NULL. IS NOT NULL evaluates to TRUE if the value is not NULL. To find the employees who do not have a department assigned, use this query:

```sql
SQL> SELECT last_name, department_id
2  FROM   employees
3  WHERE  department_id IS NULL;

LAST_NAME                 DEPARTMENT_ID
------------------------- -------------
Grant

**LIKE**

Using the LIKE operator, you can perform pattern matching. The pattern-search character % is used to match any character and any number of characters. The pattern-search character _ is used to match any single character. If you are looking for the actual character % or _ in the pattern search, you can include an escape character in the search string and notify Oracle using the ESCAPE clause.

The following query searches for all employees whose first name begins with Su and last name does not begin with S.
SQL> SELECT first_name, last_name
FROM   employees
WHERE  first_name LIKE 'Su%
AND    last_name NOT LIKE 'S%';

FIRST_NAME           LAST_NAME
-------------------- -------------------------
Sundar               Ande
Sundita              Kumar
Susan                Mavris
SQL>

The following example looks for all JOB_ID values that begin with AC_.
Since _ is a pattern-matching character, we must qualify it with an escape char-
acter. Oracle does not have a default escape character.

SQL> SELECT job_id, job_title
FROM   jobs
WHERE  job_id like 'AC\_%' ESCAPE '\';

JOB_ID     JOB_TITLE
---------- -----------------------------------
AC_MGR     Accounting Manager
AC_ACCOUNT Public Accountant
SQL>

Table 1.8 shows more examples of pattern matching.

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Matches</th>
<th>Does Not Match</th>
</tr>
</thead>
<tbody>
<tr>
<td>%SONI_1</td>
<td>SONIC1, ULTRASONI21</td>
<td>SONICS1, SONI315</td>
</tr>
<tr>
<td>_IME</td>
<td>TIME, LIME</td>
<td>IME, CRIME</td>
</tr>
<tr>
<td>%SONI_1 ESCAPE <code>'</code></td>
<td>%SONIC1, %SONI91</td>
<td>SONIC1, ULTRASONIC1</td>
</tr>
<tr>
<td>%ME__LE ESCAPE <code>'</code></td>
<td>CRIME_FILE, TIME_POLE</td>
<td>CRIMESPILE, CRIME_ALE</td>
</tr>
</tbody>
</table>
Sorting Rows

The SELECT statement may include the ORDER BY clause to sort the resulting rows in a specific order based on the data in the columns. Without the ORDER BY clause, there is no guarantee that the rows will be returned in any specific order. If an ORDER BY clause is specified, by default, the rows are returned by ascending order of the columns specified. If you need to sort the rows in descending order, use the keyword DESC next to the column name. You may specify the keyword ASC to explicitly state to sort in ascending order, although it is the default. The ORDER BY clause follows the FROM clause and WHERE clause in the SELECT statement.

To retrieve all employee names of department 90 from the EMPLOYEES table ordered by last name, use this query:

```sql
SQL> SELECT first_name || ' ' || last_name "Employee Name"
  2  FROM   employees
  3  WHERE  department_id = 90
  4* ORDER BY last_name;

Employee Name
----------------------------------------------
Lex De Haan
Steven King
Neena Kochhar
SQL>
```

You can specify more than one column in the ORDER BY clause. In this case, the result set will be ordered by the first column in the ORDER BY clause, then the second, and so on. Columns or expressions not used in the SELECT clause can also be used in the ORDER BY clause. The following example shows the use of DESC and multiple columns in the ORDER BY clause.

```sql
SQL> SELECT first_name, hire_date, salary, manager_id mid
  2  FROM   employees
  3  WHERE  department_id IN (110,100)
  4* ORDER BY mid ASC, salary DESC, hire_date;
```
You can use column alias names in the ORDER BY clause.

If the DISTINCT keyword is used in the SELECT clause, you can use only those columns listed in the SELECT clause in the ORDER BY clause. If you have used any operators on columns in the SELECT clause, the ORDER BY clause also should use them. Here is an example:

```
SQL> SELECT DISTINCT 'Region ' || region_id
2  FROM   countries
3  ORDER BY region_id;
ORDER BY region_id
*  
ERROR at line 3:  
ORA-01791: not a SELECTed expression
```

```
SQL> SELECT DISTINCT 'Region ' || region_id
2  FROM   countries
3  ORDER BY 'Region ' || region_id;
```
Basic SQL SELECT Statements

Not only can you use the column name or column alias to sort the result set of a query, you can also sort the results by specifying the position of the column in the SELECT clause. This is very useful if you have a lengthy expression in the SELECT clause and you need the results sorted on this value. The following example sorts the result set using positional values.

```
SQL> SELECT first_name, hire_date, salary, manager_id mid 2   FROM   employees 3   WHERE  department_id IN (110,100) 4* ORDER BY 4, 2, 3;
```

<table>
<thead>
<tr>
<th>FIRST_NAME</th>
<th>HIRE_DATE</th>
<th>SALARY</th>
<th>MID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shelley</td>
<td>07-JUN-94</td>
<td>12000</td>
<td>101</td>
</tr>
<tr>
<td>Nancy</td>
<td>17-AUG-94</td>
<td>12000</td>
<td>101</td>
</tr>
<tr>
<td>Daniel</td>
<td>16-AUG-94</td>
<td>9000</td>
<td>108</td>
</tr>
<tr>
<td>John</td>
<td>28-SEP-97</td>
<td>8200</td>
<td>108</td>
</tr>
<tr>
<td>Ismael</td>
<td>30-SEP-97</td>
<td>7700</td>
<td>108</td>
</tr>
<tr>
<td>Jose Manuel</td>
<td>07-MAR-98</td>
<td>7800</td>
<td>108</td>
</tr>
<tr>
<td>Luis</td>
<td>07-DEC-99</td>
<td>6900</td>
<td>108</td>
</tr>
<tr>
<td>William</td>
<td>07-JUN-94</td>
<td>8300</td>
<td>205</td>
</tr>
</tbody>
</table>

8 rows selected.

SQL>

The ORDER BY clause cannot have more than 255 columns or expressions.
Sorting NULLs

By default, in an ascending order sort, the NULL values appear at the bottom of the result set; that is, NULLs are sorted higher. For descending order sorts, NULL values appear at the top of the result set—again, NULL values are sorted higher. The default behavior can be changed by using the NULLS FIRST or NULLS LAST keywords, along with the column names (or alias names or positions). The following examples demonstrate the use of NULLS FIRST in an ascending sort.

```sql
SQL> SELECT last_name, commission_pct
    2  FROM employees
    3  WHERE last_name LIKE 'R%'
   4* ORDER BY commission_pct ASC, last_name DESC;

LAST_NAME                 COMMISSION_PCT
------------------------- --------------
Russell                               .4
Rogers
Raphaely
Rajs

SQL> SELECT last_name, commission_pct
    2  FROM employees
    3  WHERE last_name LIKE 'R%'
   4* ORDER BY commission_pct ASC NULLS FIRST, last_name DESC;

LAST_NAME                 COMMISSION_PCT
------------------------- --------------
Rogers
Raphaely
Rajs
Russell                               .4

SQL>
```
Why Do We Limit and Sort Rows?

The power of an RDBMS and SQL lies in getting exactly what we want from the database. The sample tables we considered under the HR schema are small, so even if you get all the information from the table, you can still find the specific data that you’re seeking. But what if you have a huge transaction table, with millions of rows?

You know how easy it is to look through a catalog in the library to find a particular book, or to search through an alphabetical listing to find your name. When querying a large table, make sure you know what you want.

The WHERE clause lets you query for exactly what you’re looking for. The ORDER BY clause lets you sort rows. The following steps can be used as an approach to query data from single table.

1. Know the columns of the table. You may issue the DESCRIBE command to get the column names and datatype. Understand which column has what information.

2. Pick the column names you are interested in including in the query. Use these columns in the SELECT clause.

3. Identify the column or columns where you can limit the rows or the columns that can show you only the rows of interest. Use these columns in the WHERE clause of the query, and supply the values as well as the appropriate operator.

4. If the query returns more than few rows, you may be interested in having them sorted in a particular order. Specify the column names and the sorting order in the ORDER BY clause of the query.

Let’s consider a table named PURCHASE_ORDERS. First, use the DESCRIBE command to list the columns.

SQL> DESCRIBE purchase_orders
The objective of the query is to find the completed orders that do not have any sales tax. You want to see the order number and total amount of the order. The corresponding columns that appear in the `SELECT` clause are `ORDER#` and `TOTAL_AMT`. Since you’re interested in only the rows with no sales tax in the completed orders, the columns to appear in the `WHERE` clause are `SALES_TAX` (checking for zero sales tax) and `ORD_STATUS` (checking for completeness of order, status code C). Since the query returns multiple rows, you want to order them by the order number. Notice that the `SALES_TAX` column can be `NULL`, so you want to make sure that you get all rows that have a sales tax amount of zero or `NULL`.

```sql
SELECT order#, total_amt
FROM   purchase_orders
WHERE  ord_status = 'C'
AND    (sales_tax IS NULL
OR    sales_tax = 0)
ORDER BY order#;
```

An alternative is to use the `NVL` function to deal with the `NULL` values. This function is discussed in Chapter 3.
Using Expressions

An expression is a combination of one or more values, operators, and SQL functions that result in a value. The result of an expression generally assumes the datatype of its components. The simple expression 5+6 evaluates to 11 and assumes a datatype of NUMBER. Expressions can appear in the following clauses:

- The SELECT clause of queries
- The WHERE clause, ORDER BY clause, and HAVING clause
- The VALUES clause of the INSERT statement
- The SET clause of the UPDATE statement

We will review the syntax of using these statements in later chapters.

You can include parentheses to group and evaluate expressions, and then apply the result to the rest of the expression. When parentheses are used, the expression in the innermost parentheses is evaluated first. Here is an example of a compound expression: 
\[(2*4)/(3+1))*10\]. The result of 2*4 is divided by the result of 3+1. Then the result from the division operation is multiplied by 10.

The **CASE Expression**

The CASE expression is new to Oracle9i and can be used to derive the IF...THEN...ELSE logic in SQL. Here is the syntax of the simple CASE expression:

```sql
CASE <expression>
WHEN <compare value> THEN <return value> … … …
[ELSE <return value>]
END
```

The CASE expression begins with the keyword CASE and ends with the keyword END. The ELSE clause is optional, the WHEN clause can be repeated for 128 times. The following query displays a description for the REGION_ID column based on the value.

```sql
SQL> SELECT country_name, region_id,
       2       CASE region_id WHEN 1 THEN 'Europe'
       3                                       WHEN 2 THEN 'America'
       4                                       WHEN 3 THEN 'Asia'
       5                                       ELSE 'Other' END Continent
       6   FROM   countries
       7* WHERE  country_name LIKE 'I%';
```
The other form of the `CASE` expression is the searched `CASE`, where the values are derived based on a condition. This version has the following syntax:

```
CASE
  WHEN <condition> THEN <return value> ... ...
[ELSE <return value>] END
```

The following example categorizes the salary as Low, Medium, and High using a searched `CASE` expression.

```sql
SQL> SELECT first_name, department_id, salary,
           CASE WHEN salary < 6000 THEN 'Low'
                WHEN salary < 10000 THEN 'Medium'
                WHEN salary >= 10000 THEN 'High' END
      FROM employees
      WHERE department_id <= 30
      ORDER BY first_name;
```

```
FIRST_NAME           DEPARTMENT_ID     SALARY CATEGOR
-------------------- ------------- ---------- -----
Alexander                       30       3100 Low
Den                             30      11000 High
Guy                             30       2600 Low
Jennifer                        10       4400 Low
Karen                           30       2500 Low
Michael                         20      13000 High
Pat                             20       6000 Medium
Shelli                          30       2900 Low
Sigal                           30       2800 Low
```

9 rows selected.

SQL>
Summary

Data in the Oracle database is managed and accessed using SQL. A SELECT statement is used to query data from a table or view. You can limit the rows selected by using a WHERE clause and order the retrieved data using the ORDER BY clause.

In this chapter, we reviewed fundamentals of SQL, including datatypes and operators. The CHAR and VARCHAR2 datatypes are used to store alphanumeric information. The NUMBER datatype is used to store any numeric value. Date values can be stored using the DATE datatype. Oracle has a wide range of operators: arithmetic, concatenation, set, comparison, membership, logical, pattern matching, range, and existence and NULL checking.

The CASE expression is new to Oracle9i. It is used to bring conditional logic to SQL.

Exam Essentials

Understand the operators.  Know the various operators that can be used in queries. The parentheses around an expression change the precedence of the operators.

Know how to execute a SQL statement.  You can execute a SQL statement by ending the statement with a semicolon, and in SQL*Plus, by having the / on a line by itself or by using the RUN command.

Understand the WHERE clause.  The WHERE clause specifies a condition to limit the number or rows returned. You cannot use column alias names in this clause.

Understand the ORDER BY clause.  The ORDER BY clause is used to sort the result set from a query. You can specify ascending order or descending order for the sort. Ascending order is the default.

Know the order of clauses in the SELECT statement.  The SELECT statement must have a FROM clause. The WHERE clause, if it exists, should follow the FROM clause and precede the ORDER BY clause.
Know the use of the DUAL table. The DUAL table is a dummy table in Oracle with one column and one row. This table is commonly used to get the values of system variables such as SYSDATE or USER.

Know the characters used for pattern matching. The % character is used to match zero or more characters. The _ character is used to match one, and only one, character. The SQL operator used with pattern-matching character is LIKE.

Key Terms

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

- unary operator
- arithmetic operators
- binary operators
- CASE
- CHAR
- column alias
- comparison operators
- concatenation operator
- DATE
- DISTINCT
- DUAL table
- escape character
- expression
- literals
- logical operators
- NUMBER
- operator
- order of precedence
- precision
- query
- scale
- seed database
- SELECT
- set operators
- SYSDATE
- VARCHAR2
- WHERE
The following table summarizes the commands used in this chapter.

<table>
<thead>
<tr>
<th>Command</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>SELECT * FROM <code>&lt;table name&gt;</code></td>
<td>Used to query all the columns and all rows of the table</td>
</tr>
<tr>
<td>SELECT <code>&lt;column&gt;</code>, <code>&lt;columns&gt;</code> FROM <code>&lt;table name&gt;</code></td>
<td>Used to query selected columns and all rows from the table</td>
</tr>
<tr>
<td>SELECT <code>&lt;column&gt;</code> FROM <code>&lt;table name&gt;</code> WHERE <code>&lt;column&gt; = &lt;value&gt;</code></td>
<td>Used to query selected columns and to restrict rows that satisfy <code>&lt;value&gt;</code> for the <code>&lt;column&gt;</code></td>
</tr>
<tr>
<td>SELECT <code>&lt;column&gt;</code>, <code>&lt;columns&gt;</code> FROM <code>&lt;table name&gt;</code> WHERE <code>&lt;column&gt; = &lt;value&gt;</code> ORDER BY <code>&lt;column&gt;</code></td>
<td>Used to query selected columns and restrict rows with result set sorted</td>
</tr>
</tbody>
</table>
Review Questions

1. You issue the following query:

   SELECT salary "Employee Salary"
   FROM employees;

   How will the column heading appear in the result?
   
   A. EMPLOYEE SALARY
   B. EMPLOYEE_SALARY
   C. Employee Salary
   D. employee_salary

2. The EMP table is defined as follows:

<table>
<thead>
<tr>
<th>Column</th>
<th>Datatype</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>EMPNO</td>
<td>NUMBER</td>
<td>4</td>
</tr>
<tr>
<td>ENAME</td>
<td>VARCHAR2</td>
<td>30</td>
</tr>
<tr>
<td>SALARY</td>
<td>NUMBER</td>
<td>14,2</td>
</tr>
<tr>
<td>COMM</td>
<td>NUMBER</td>
<td>10,2</td>
</tr>
<tr>
<td>DEPTNO</td>
<td>NUMBER</td>
<td>2</td>
</tr>
</tbody>
</table>

   You perform the following two queries:

   1. SELECT empno number, ename
      FROM emp ORDER BY 1;
   2. SELECT empno, ename
      FROM emp ORDER BY empno ASC;

   Which of the following is true?
   
   A. Statements 1 and 2 will produce the same result.
   B. Statement 1 will execute; statement 2 will return an error.
   C. Statement 2 will execute; statement 1 will return an error.
   D. Statements 1 and 2 will execute but produce different results.
3. You issue the following SELECT statement on the EMP table shown in question 2.

   SELECT (200+((salary*0.1)/2)) FROM emp;

   What will happen to the result if all of the parentheses are removed?
   A. No difference, because the answer will always be NULL.
   B. No difference, because the result will be the same.
   C. The result will be higher.
   D. The result will be lower.

4. In the following SELECT statement, which component is a literal? (Choose all that apply.)

   SELECT 'Employee Name: ' || ename
   FROM emp where deptno = 10;
   A. 10
   B. ename
   C. Employee Name:
   D. ||

5. When you try to save 34567.2255 into a column defined as NUMBER(7,2) what value is actually saved?
   A. 34567.00
   B. 34567.23
   C. 34567.22
   D. 3456.22

6. What is the default display length of the DATE datatype column?
   A. 8
   B. 9
   C. 19
   D. 6
7. What will happen if you query the EMP table shown in question 2 with the following?

```sql
SELECT empno, DISTINCT ename, salary FROM emp;
```

A. EMPNO, unique values of ENAME and then SALARY are displayed.

B. EMPNO, unique values of the two columns, ENAME and SALARY, are displayed.

C. DISTINCT is not a valid keyword in SQL.

D. No values will be displayed because the statement will return an error.

8. Which clause in a query limits the rows selected?

A. ORDER BY

B. WHERE

C. SELECT

D. FROM

9. The following listing shows the records of the EMP table.

<table>
<thead>
<tr>
<th>EMPNO</th>
<th>ENAME</th>
<th>SALARY</th>
<th>COMM</th>
<th>DEPTNO</th>
</tr>
</thead>
<tbody>
<tr>
<td>7369</td>
<td>SMITH</td>
<td>800</td>
<td></td>
<td>20</td>
</tr>
<tr>
<td>7499</td>
<td>ALLEN</td>
<td>1600</td>
<td>300</td>
<td>30</td>
</tr>
<tr>
<td>7521</td>
<td>WARD</td>
<td>1250</td>
<td>500</td>
<td>30</td>
</tr>
<tr>
<td>7566</td>
<td>JONES</td>
<td>2975</td>
<td></td>
<td>20</td>
</tr>
<tr>
<td>7654</td>
<td>MARTIN</td>
<td>1250</td>
<td>1400</td>
<td>30</td>
</tr>
<tr>
<td>7698</td>
<td>BLAKE</td>
<td>2850</td>
<td></td>
<td>30</td>
</tr>
<tr>
<td>7782</td>
<td>CLARK</td>
<td>2450</td>
<td>24500</td>
<td>10</td>
</tr>
<tr>
<td>7788</td>
<td>SCOTT</td>
<td>3000</td>
<td></td>
<td>20</td>
</tr>
<tr>
<td>7839</td>
<td>KING</td>
<td>5000</td>
<td>50000</td>
<td>10</td>
</tr>
<tr>
<td>7844</td>
<td>TURNER</td>
<td>1500</td>
<td>0</td>
<td>30</td>
</tr>
<tr>
<td>7876</td>
<td>ADAMS</td>
<td>1100</td>
<td></td>
<td>20</td>
</tr>
<tr>
<td>7900</td>
<td>JAMES</td>
<td>950</td>
<td></td>
<td>30</td>
</tr>
<tr>
<td>7902</td>
<td>FORD</td>
<td>3000</td>
<td></td>
<td>20</td>
</tr>
<tr>
<td>7934</td>
<td>MILLER</td>
<td>1300</td>
<td>13000</td>
<td>10</td>
</tr>
</tbody>
</table>
When you issue the following query, which value will be displayed in the first row?

```
SELECT empno
FROM emp
WHERE deptno = 10
ORDER BY ename DESC;
```

A. MILLER  
B. 7934  
C. 7876  
D. No rows will be returned because ename cannot be used in the ORDER BY clause.

10. Refer to the listing of records in the EMP table in question 9. How many rows will the following query return?

```
SELECT * FROM emp WHERE ename BETWEEN 'A' AND 'C'
```

A. 4  
B. 2  
C. A character column cannot be used in the BETWEEN operator.  
D. 3

11. Refer to the EMP table in question 2. When you issue the following query, which line has an error?

```
SELECT empno "Enumber", ename "EmpName"
FROM emp
WHERE deptno = 10
AND  "Enumber" = 7782
ORDER BY "Enumber";
```

A. 1  
B. 5  
C. 4  
D. No error; the statement will finish successfully.
12. You issue the following query:

```sql
SELECT empno, ename
FROM emp
WHERE empno = 7782 OR empno = 7876;
```

Which other operator can replace the OR condition in the `WHERE` clause?

A. IN  
B. BETWEEN .. AND ..  
C. LIKE  
D. <=  
E. >=

13. The following are clauses of the SELECT statement:

1. WHERE  
2. FROM  
3. ORDER BY

In which order should they appear in a query?

A. 1, 3, 2  
B. 2, 1, 3  
C. 2, 3, 1  
D. The order of these clauses does not matter.

14. Which statement searches for PRODUCT_ID values that begin with DI_ from the ORDERS table?

A. SELECT * FROM ORDERS
   WHERE PRODUCT_ID = 'DI%';
B. SELECT * FROM ORDERS
   WHERE PRODUCT_ID LIKE 'DI_' ESCAPE '\';
C. SELECT * FROM ORDERS
   WHERE PRODUCT_ID LIKE 'DI\_%' ESCAPE '\';
D. SELECT * FROM ORDERS
   WHERE PRODUCT_ID LIKE 'DI\_' ESCAPE '\';
E. SELECT * FROM ORDERS
   WHERE PRODUCT_ID LIKE 'DI_\%' ESCAPE '\';
15. COUNTRY_NAME and REGION_ID are valid column names in the COUNTRIES table. Which one of the following statements will execute without an error?

A. SELECT country_name, region_id,
   CASE region_id = 1 THEN 'Europe',
   region_id = 2 THEN 'America',
   region_id = 3 THEN 'Asia',
   ELSE 'Other' END Continent
   FROM   countries;

B. SELECT country_name, region_id,
   CASE (region_id WHEN 1 THEN 'Europe',
   WHEN 2 THEN 'America',
   WHEN 3 THEN 'Asia',
   ELSE 'Other') Continent
   FROM   countries;

C. SELECT country_name, region_id,
   CASE region_id WHEN 1 THEN 'Europe'
   WHEN 2 THEN 'America'
   WHEN 3 THEN 'Asia'
   ELSE 'Other' END Continent
   FROM   countries;

D. SELECT country_name, region_id,
   CASE region_id WHEN 1 THEN 'Europe'
   WHEN 2 THEN 'America'
   WHEN 3 THEN 'Asia'
   ELSE 'Other' Continent
   FROM   countries;
16. Which special character is used to query all the columns from the table without listing each column by name?

   A. %
   B. &
   C. @
   D. *

17. The EMPLOYEE table has the following data:

<table>
<thead>
<tr>
<th>EMP_NAME</th>
<th>HIRE_DATE</th>
<th>SALARY</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMITH</td>
<td>17-DEC-90</td>
<td>800</td>
</tr>
<tr>
<td>ALLEN</td>
<td>20-FEB-91</td>
<td>1600</td>
</tr>
<tr>
<td>WARD</td>
<td>22-FEB-91</td>
<td>1250</td>
</tr>
<tr>
<td>JONES</td>
<td>02-APR-91</td>
<td>5975</td>
</tr>
<tr>
<td>WARDEN</td>
<td>28-SEP-91</td>
<td>1250</td>
</tr>
<tr>
<td>BLAKE</td>
<td>01-MAY-91</td>
<td>2850</td>
</tr>
</tbody>
</table>

What will be the value in the first row of the result set when the following query is executed?

```
SELECT hire_date FROM employee
ORDER BY salary, emp_name;
```

A. 02-APR-91
B. 17-DEC-90
C. 28-SEP-91
D. The query is invalid, because you cannot have a column in the ORDER BY clause that is not part of the SELECT clause.
18. Which SQL statement will query the EMPLOYEES table for FIRST_NAME, LAST_NAME, and SALARY of all employees in DEPARTMENT_ID 40 in the alphabetical order of last name?

A. SELECT first_name last_name salary 
   FROM employees 
   ORDER BY last_name 
   WHERE department_id = 40;
B. SELECT first_name, last_name, salary 
   FROM employees 
   ORDER BY last_name ASC 
   WHERE department_id = 40;
C. SELECT first_name last_name salary 
   FROM employees 
   WHERE department_id = 40 
   ORDER BY last_name ASC;
D. SELECT first_name, last_name, salary 
   FROM employees 
   WHERE department_id = 40 
   ORDER BY last_name;
E. SELECT first_name, last_name, salary 
   FROM TABLE employees 
   WHERE department_id IS 40 
   ORDER BY last_name ASC;

19. When doing pattern matching using the LIKE operator, which character is used as the default escape character by Oracle?

A. | 
B. / 
C. \ 
D. There is no default escape character in Oracle9i.

20. Column alias names cannot be used in which clause?

A. SELECT clause 
B. WHERE clause 
C. ORDER BY clause 
D. None of the above
Answers to Review Questions

1. C. Column alias names enclosed in quotation marks will appear as typed. Spaces and mixed case appear in the column alias name only when the alias is enclosed in double quotation marks.

2. A. Statements 1 and 2 will produce the same result. You can use the column name, column alias, or column position in the ORDER BY clause. The default sort order is ascending. For a descending sort, you must explicitly specify that order with the DESC keyword.

3. B. In the arithmetic evaluation, multiplication and division have precedence over addition and subtraction. Even if you do not include the parentheses, salary*0.1 will be evaluated first. The result is then divided by 2, and its result is added to 200.

4. A, C. Character literals in the SQL statement are enclosed in single quotation marks. Literals are concatenated using ||. Employee Name: is a character literal, and 10 is a numeric literal.

5. B. Since the numeric column is defined with precision 7 and scale 2, you can have five digits in the integer part and two digits after the decimal point. The digits after the decimal are rounded.

6. B. The default display format of the DATE column is DD-MON-YY, whose length is 9. This is U.S. specific and will be different as user settings vary.

7. D. DISTINCT is used to display a unique result row, and it should follow immediately after the keyword SELECT. Uniqueness is identified across the row, not a single column.

8. B. The WHERE clause is used to limit the rows returned from a query. The WHERE clause condition is evaluated, and rows are returned only if the result is TRUE. The ORDER BY clause is used to display the result in certain order.
9. B. There are three records belonging to DEPTNO 10: EMPNO 7934 (MILLER), 7839 (KING), and 7782 (CLARK). When you sort their names by descending order, MILLER is the first row to display. You can use alias names and columns that are not in the SELECT clause in the ORDER BY clause.

10. D. Here, a character column is compared against a string using the BETWEEN operator, which is equivalent to ename >= 'A' AND ename <= 'C'. The name CLARK will not be included in this query, because 'CLARK' is > 'C'.

11. C. Column alias names cannot be used in the WHERE clause. They can be used in the ORDER BY clause.

12. A. The IN operator can be used. You can write the WHERE clause as WHERE empno IN (7782, 7876);

13. B. The FROM clause appears after the SELECT statement, followed by WHERE and ORDER BY clauses. The FROM clause specifies the table names, the WHERE clause limits the result set, and the ORDER BY clause sorts the result.

14. C. Since _ is a special pattern-matching character, you need to include the ESCAPE clause in LIKE. The % character matches any number of characters including 0, and _ matches a single character.

15. C. A CASE expression begins with the keyword CASE and ends with keyword END.

16. D. An asterisk (*) is used to denote all columns in a table.

17. B. The default sorting order for numeric column is ascending. The columns are sorted first by salary and then by name, so the row with the lowest salary is displayed first. It is perfectly valid to use a column in the ORDER BY clause that is not part of the SELECT clause.
18. D. In the SELECT clause, the column names should be separated by commas. An alias name may be provided for each column with a space or using the keyword AS. The FROM clause should appear after the SELECT clause. The WHERE clause appears after the FROM clause. The ORDER BY clause comes after the WHERE clause.

19. D. There is no default escape character in Oracle9i. If your search includes pattern-matching characters such as _ or %, define an escape character using the ESCAPE keyword in the LIKE operator.

20. B. Column alias names cannot be used in the WHERE clause of the SQL statement. In the ORDER BY clause, you can use the column name or alias name, or indicate the column by its position in the SELECT clause.