CHAPTER 4

SINGLE-TABLE QUERIES

LEARNING OBJECTIVES

Objectives

- Retrieve data from a database using SQL commands
- Use simple and compound conditions in queries
- Use the BETWEEN, LIKE, and IN operators in queries
- Use computed columns in queries
- Sort data using the ORDER BY clause
- Sort data using multiple keys and in ascending and descending order
- Use aggregate functions in a query
- Use subqueries
- Group data using the GROUP BY clause
- Select individual groups of data using the HAVING clause
- Retrieve columns with null values

INTRODUCTION

In this chapter, you will learn about the SQL SELECT command that is used to retrieve data in a database. You will examine ways to sort data and use SQL functions to count rows and calculate totals. You also will learn how to nest SELECT commands by placing one SELECT command inside another. Finally, you will learn how to group rows that have matching values in some column.

CONSTRUCTING SIMPLE QUERIES

One of the most important features of a DBMS is its ability to answer a wide variety of questions concerning the data in a database. When you need to find data that answers a specific question, you use a query. A **query** is a question represented in a way that the DBMS can understand.

In SQL, you use the SELECT command to query a database. The basic form of the SELECT command is SELECT-FROM-WHERE. After you type the word SELECT, you list the columns that you want to include in the query results. This portion of the command is called the **SELECT clause**. Next, you type the word FROM followed by the name of the table that contains the data you need to query. This portion of the command is called the **FROM clause**. Finally, after the word WHERE, you list any conditions (restrictions) that apply to the data you want to retrieve. This optional portion of the command is called the **WHERE clause**. For example, when you need to retrieve the rows for only those customers with credit limits of $7,500, include a condition in the WHERE clause specifying that the value in the CREDIT_LIMIT column must be $7,500 (CREDIT_LIMIT = 7500).

There are no special formatting rules in SQL. In this text, the FROM clause and the WHERE clause (when it is used) appear on separate lines only to make the commands more readable and understandable.

Retrieving Certain Columns and All Rows

You can write a command to retrieve specified columns and all rows from a table, as illustrated in Example 1.

EXAMPLE 1

List the number, name, and balance for all customers.

Because you need to list *all* customers, you do not need to include a WHERE clause; you do not need to put any restrictions on the data to retrieve. You simply list the columns to be included (CUSTOMER_NUM, CUSTOMER_NAME, and BALANCE) in the SELECT clause and the name of the table (CUSTOMER) in the FROM clause. Type a semicolon to indicate the end of the command, and then click the Run button to display the results. The query and its results appear in Figure 4-1.

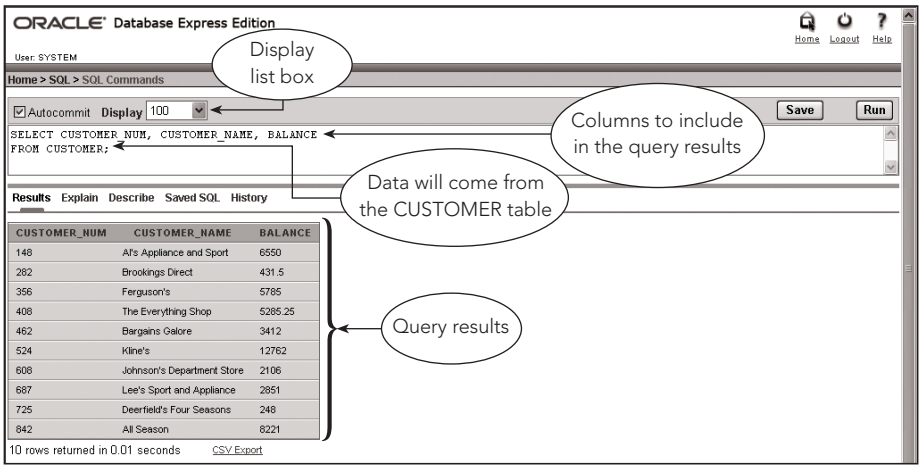


FIGURE 4-1 SELECT command to select certain columns from the CUSTOMER table

NOTE

In the Oracle Database Express Edition, the number in the Display list box indicates the maximum number of rows that Oracle will display in the query results. The default value is 10. To change the value, either click the arrow and select a new value from the list or type a new value in the box. Figure 4-1 shows the Display list box after the user changed it to display 100 rows. When you run a query whose results will include more rows than the number in the Display list box, Oracle will display a message indicating this fact. If this situation occurs, increase the number in the Display list box, and then click the Run button again to display the complete query results.

NOTE

If you are using Access or SQL Server to run the SQL commands shown in this text, your query results will differ slightly from the results shown in the figures. In Access, the BALANCE field has the CURRENCY data type and Access will display values in this column with two decimal places and a dollar sign. In SQL Server, values in the BALANCE field will be displayed with two decimal places and DATE field values might be displayed with a time value. Although your output might be formatted differently, the data should be the same as what you see in the figures.

Retrieving All Columns and All Rows

You can use the same type of command illustrated in Example 1 to retrieve all columns and all rows from a table. As Example 2 illustrates, however, you can use a shortcut to accomplish this task.

EXAMPLE 2

List the complete PART table.

Instead of including every column in the SELECT clause, you can use an asterisk (*) to indicate that you want to include all columns. The result lists all columns in the order in which you described them to the DBMS when you created the table. If you want the columns listed in a different order, type the column names in the order in which you want them to appear in the query results. In this case, assuming that the default order is appropriate, you can use the query shown in Figure 4-2 to display the complete PART table.

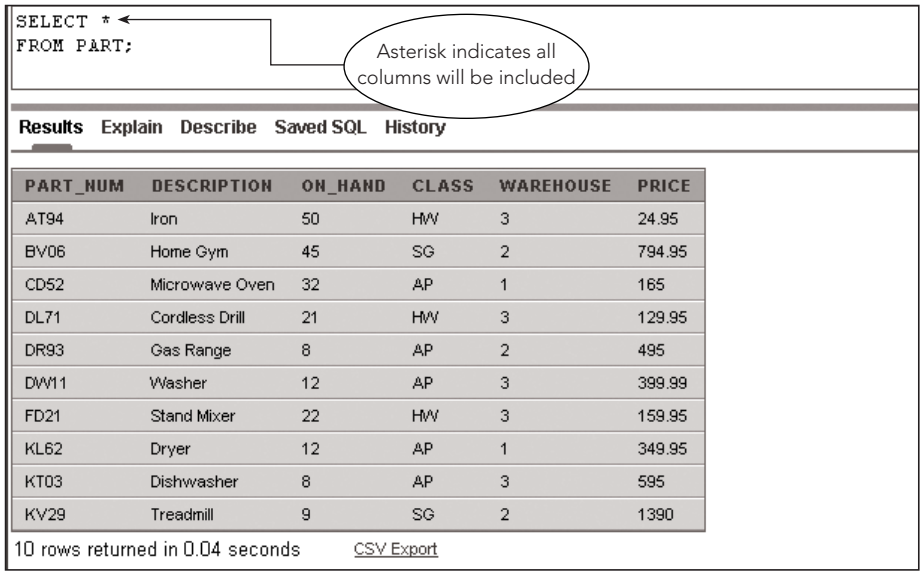


FIGURE 4-2 SELECT command to select all columns from the PART table

Using a WHERE Clause

When you need to retrieve rows that satisfy some condition, you include a WHERE clause in the SELECT command, as shown in Example 3.

EXAMPLE 3

What is the name of the customer with customer number 148?

You can use a WHERE clause to restrict the query results to customer number 148, as shown in Figure 4-3. Because CUSTOMER_NUM is a character column, the value 148 is enclosed in single quotation marks. In addition, because the CUSTOMER_NUM column is the primary key of the CUSTOMER table, there can be only one customer whose number matches the number in the WHERE clause.

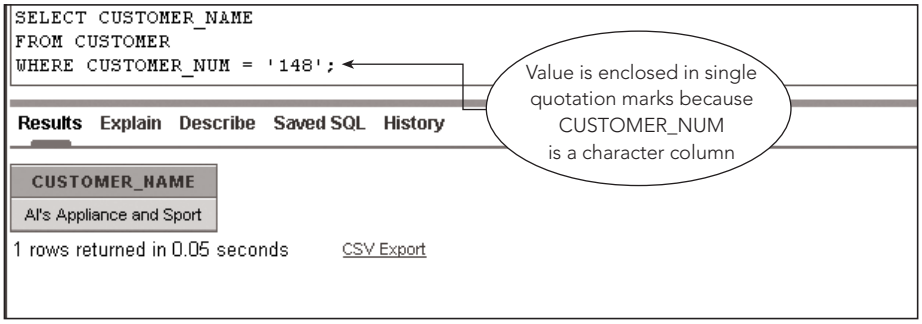


FIGURE 4-3 SELECT command to find the name of customer number 148

The condition in the preceding WHERE clause is called a simple condition. A **simple condition** has the form column name, comparison operator, and then either another column name or a value. Figure 4-4 lists the comparison operators that you can use in SQL. Notice that there are two versions of the "not equal to" operator: < > and !=.

Comparison operator	Description
=	Equal to
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
< >	Not equal to
!=	Not equal to

FIGURE 4-4 Comparison operators used in SQL commands

EXAMPLE 4

Find the number and name of each customer located in the city of Grove.

The only difference between this example and the previous one is that in Example 3, there could not be more than one row in the answer because the condition involved the table's primary key. In Example 4, the condition involves a column that is *not* the table's primary key. Because there is more than one customer located in the city of Grove, the results can and do contain more than one row, as shown in Figure 4-5.

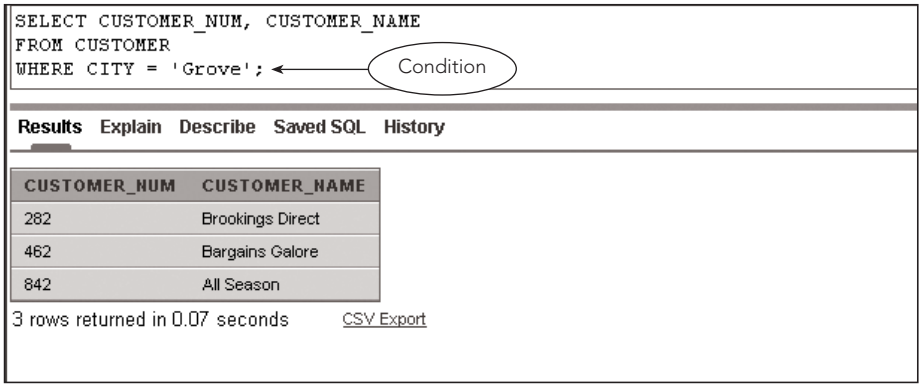


FIGURE 4-5 SELECT command to find all customers located in Grove

EXAMPLE 5

Find the number, name, balance, and credit limit for all customers with balances that exceed their credit limits.

A simple condition can also compare the values stored in two columns. In Figure 4-6, the WHERE clause includes a comparison operator that selects only those rows in which the balance is greater than the credit limit.

```
SELECT CUSTOMER_NUM, CUSTOMER_NAME, BALANCE, CREDIT_LIMIT
FROM CUSTOMER
WHERE BALANCE > CREDIT_LIMIT;
```

Results Explain Describe Saved SQL History

CUSTOMER_NUM	CUSTOMER_NAME	BALANCE	CREDIT_LIMIT
408	The Everything Shop	5285.25	5000
842	All Season	8221	7500

2 rows returned in 0.03 seconds CSV Export

FIGURE 4-6 SELECT command to find all customers with balances that exceed their credit limits

Using Compound Conditions

The conditions you have seen so far are called simple conditions. The following examples require compound conditions. You form a **compound condition** by connecting two or more simple conditions with the AND, OR, and NOT operators. When the **AND** operator connects simple conditions, all the simple conditions must be true in order for the compound condition to be true. When the **OR** operator connects the simple conditions, the compound condition will be true whenever any one of the simple conditions is true. Preceding a condition by the **NOT** operator reverses the truth of the original condition. For example, if the original condition is true, the new condition will be false; if the original condition is false, the new one will be true.

EXAMPLE 6

List the descriptions of all parts that are located in warehouse 3 and for which there are more than 25 units on hand.

In Example 6, you need to retrieve those parts that meet *both* conditions—the warehouse number is equal to 3 *and* the number of units on hand is greater than 25. To find the answer, you form a compound condition using the AND operator, as shown in Figure 4-7. The query examines the data in the PART table and lists the parts that are located in warehouse 3 and for which there are more than 25 units on hand. When a WHERE clause uses the AND operator to connect simple conditions, it also is called an **AND condition**.

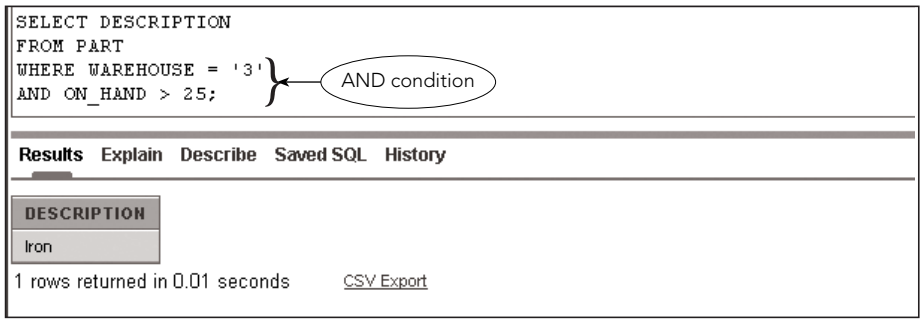


FIGURE 4-7 SELECT command with an AND condition on separate lines

For readability, each of the simple conditions in the query shown in Figure 4-7 appears on a separate line. Some people prefer to put the conditions on the same line with parentheses around each simple condition, as shown in Figure 4-8. These two methods accomplish the same thing. In this text, simple conditions will appear on separate lines and without parentheses.

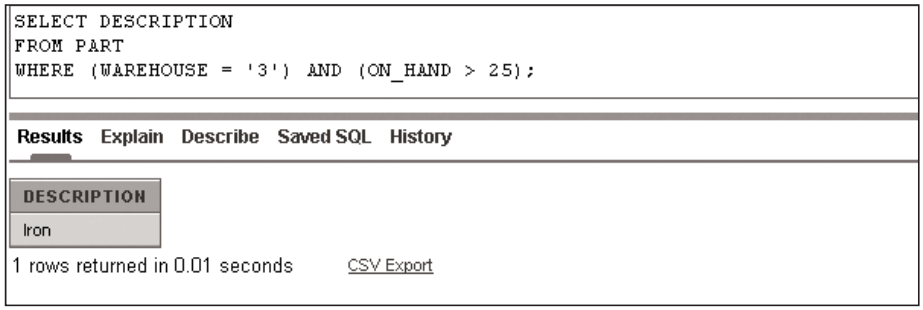


FIGURE 4-8 SELECT command with an AND condition on a single line

EXAMPLE 7

List the descriptions of all parts that are located in warehouse 3 or for which there are more than 25 units on hand.

In Example 7, you need to retrieve descriptions for those parts for which the warehouse number is equal to 3, *or* the number of units on hand is greater than 25, *or* both. To do this, you form a compound condition using the OR operator, as shown in Figure 4-9. When a WHERE clause uses the OR operator to connect simple conditions, it also is called an **OR condition**.

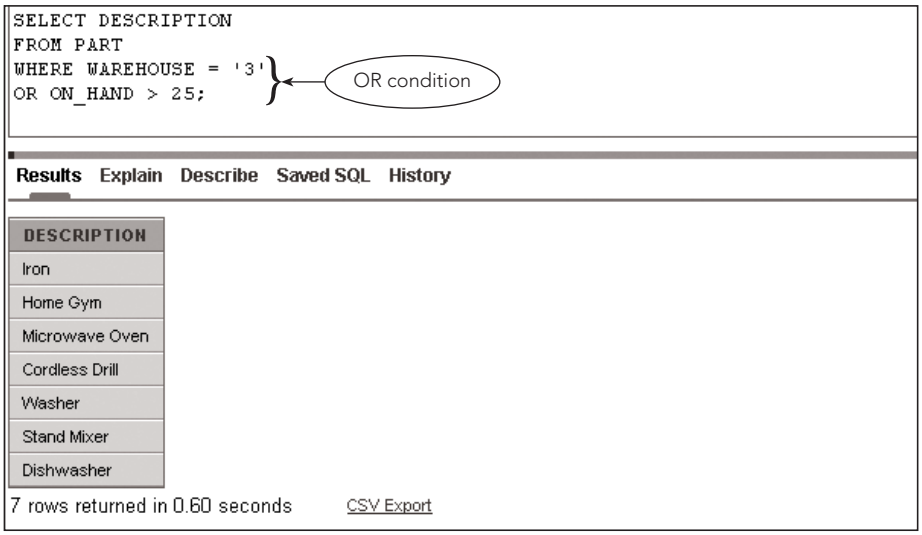


FIGURE 4-9 SELECT command with an OR condition

EXAMPLE 8

List the descriptions of all parts that are not in warehouse 3.

For Example 8, you could use a simple condition and the "not equal to" operator (WHERE WAREHOUSE < > '3'). As an alternative, you could use the EQUAL operator (=) in the condition and precede the entire condition with the NOT operator, as shown in Figure 4-10. When a WHERE clause uses the NOT operator to connect simple conditions, it also is called a **NOT condition**.

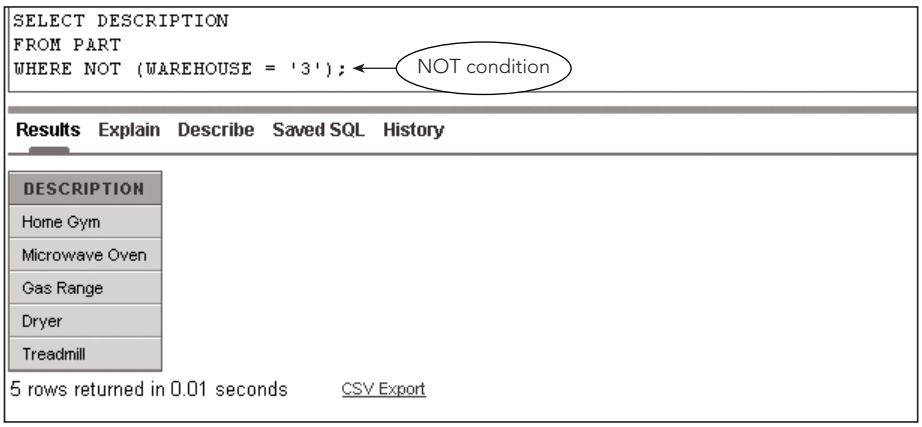


FIGURE 4-10 SELECT command with a NOT condition

You do not need to enclose the condition WAREHOUSE = '3' in parentheses, but doing so makes the command more readable.

Using the BETWEEN Operator

Example 9 requires a compound condition to determine the answer.

EXAMPLE 9

List the number, name, and balance of all customers with balances greater than or equal to $2,000 and less than or equal to $5,000.

You can use a WHERE clause and the AND operator, as shown in Figure 4-11, to retrieve the data.

```
SELECT CUSTOMER_NUM, CUSTOMER_NAME, BALANCE
FROM CUSTOMER
WHERE BALANCE >= 2000
AND BALANCE <= 5000;
```

Results Explain Describe Saved SQL History

CUSTOMER_NUM	CUSTOMER_NAME	BALANCE
462	Bargains Galore	3412
608	Johnson's Department Store	2106
687	Lee's Sport and Appliance	2851

3 rows returned in 0.20 seconds CSV Export

FIGURE 4-11 SELECT command with an AND condition for a single column

NOTE

In SQL, numbers included in queries are entered without extra symbols, such as dollar signs and commas.

An alternative to this approach uses the BETWEEN operator, as shown in Figure 4-12. The **BETWEEN** operator lets you specify a range of values in a condition.

```
SELECT CUSTOMER_NUM, CUSTOMER_NAME, BALANCE
FROM CUSTOMER
WHERE BALANCE BETWEEN 2000 AND 5000;
```

Results Explain Describe Saved SQL History

CUSTOMER_NUM	CUSTOMER_NAME	BALANCE
462	Bargains Galore	3412
608	Johnson's Department Store	2106
687	Lee's Sport and Appliance	2851

3 rows returned in 0.00 seconds CSV Export

FIGURE 4-12 SELECT command with the BETWEEN operator

The BETWEEN operator is inclusive, meaning that the query selects a value equal to either value in the condition and in the range of the values. In the clause BETWEEN 2000 and 5000, for example, values of 2,000 through 5,000 would make the condition true. You can use the BETWEEN operator in Oracle, SQL Server, and Access.

The BETWEEN operator is not an essential feature of SQL; you have just seen that you can obtain the same result without it. Using the BETWEEN operator, however, does make certain SELECT commands simpler to construct.

Using Computed Columns

You can perform computations using SQL queries. A **computed column** does not exist in the database but can be computed using data in the existing columns. Computations can involve any arithmetic operator shown in Figure 4-13.

Arithmetic operator	Description
+	Addition
–	Subtraction
*	Multiplication
/	Division

FIGURE 4-13 Arithmetic operators

EXAMPLE 10

Find the number, name, and available credit (the credit limit minus the balance) for each customer.

There is no column in the Premiere Products database that stores a customer's available credit, but you can compute the available credit using the CREDIT_LIMIT and BALANCE columns. To compute the available credit, you use the expression CREDIT_LIMIT - BALANCE, as shown in Figure 4-14.

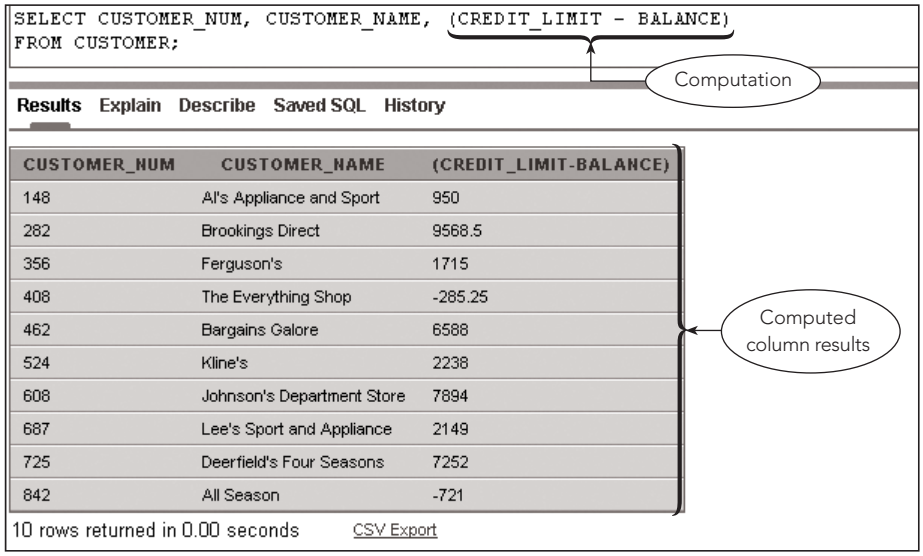


FIGURE 4-14 SELECT command with a computed column

The parentheses around the calculation (CREDIT_LIMIT - BALANCE) are not essential but improve readability.

You also can assign a name to a computed column by following the computation with the word AS and the desired name. The command shown in Figure 4-15, for example, assigns the name AVAILABLE_CREDIT to the computed column.

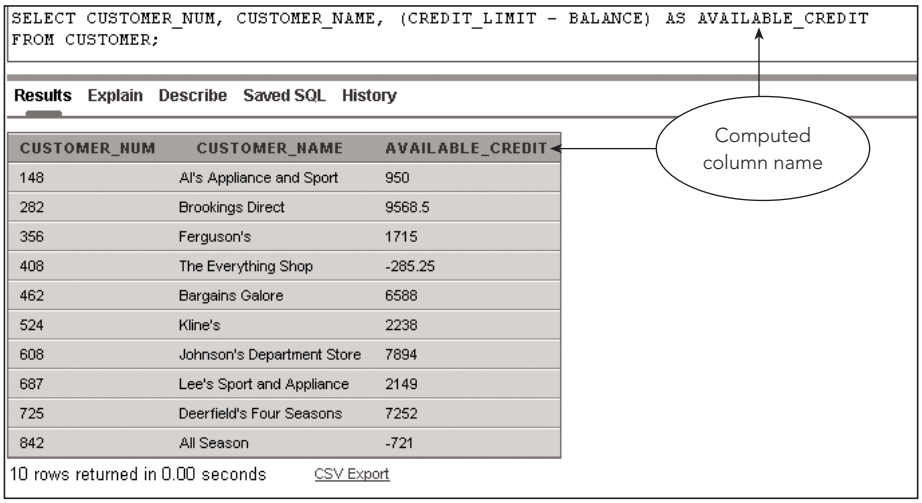


FIGURE 4-15 SELECT command with a named computed column

NOTE

You can use names containing spaces following the word AS. In many SQL implementations, including Oracle, you do so by enclosing the name in quotation marks (for example, AS "AVAILABLE CREDIT"). Other SQL implementations require you to enclose the name in other special characters. For example, in Access you would enclose the name in square brackets (AS [AVAILABLE CREDIT]). In SQL Server, you can use either quotation marks or square brackets.

EXAMPLE 11

Find the number, name, and available credit for each customer with at least $5,000 of available credit.

You also can use computed columns in comparisons, as shown in Figure 4-16.

```
SELECT CUSTOMER_NUM, CUSTOMER_NAME, (CREDIT_LIMIT - BALANCE) AS AVAILABLE_CREDIT
FROM CUSTOMER
WHERE (CREDIT_LIMIT - BALANCE) >= 5000;
```

Results Explain Describe Saved SQL History

CUSTOMER_NUM	CUSTOMER_NAME	AVAILABLE_CREDIT
282	Brookings Direct	9568.5
462	Bargains Galore	6588
608	Johnson's Department Store	7894
725	Deerfield's Four Seasons	7252

4 rows returned in 0.02 seconds CSV Export

FIGURE 4-16 SELECT command with a computation in the condition

Using the LIKE Operator

In most cases, the conditions in WHERE clauses involve exact matches, such as retrieving rows for each customer located in the city of Grove. In some cases, however, exact matches do not work. For example, you might know that the desired value contains only a certain collection of characters. In such cases, you use the LIKE operator with a wildcard symbol, as shown in Example 12. Rather than testing for equality, the **LIKE** operator uses one or more wildcard characters to test for a pattern match.

EXAMPLE 12

List the number, name, and complete address of each customer located on a street that contains the letters "Central."

All you know is that the addresses you want contain a certain collection of characters ("Central") somewhere in the STREET column, but you do not know where. In SQL for Oracle and for SQL Server, the percent sign (%) is used as a wildcard to represent any collection of characters. As shown in Figure 4-17, the condition LIKE '%Central%' retrieves information for each customer whose street contains some collection of characters, followed by the letters "Central," followed potentially by some additional characters. Note that this query also would retrieve information for a customer whose address is "123 Centralia" because "Centralia" also contains the letters "Central."

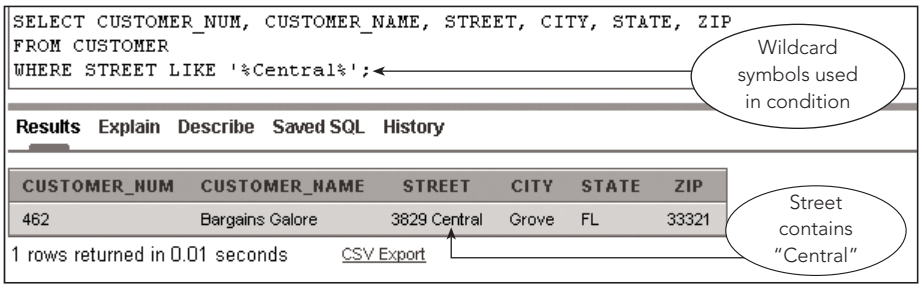


FIGURE 4-17 SELECT command with a LIKE operator and wildcards

Another wildcard symbol in SQL is the underscore (_), which represents any individual character. For example, "T_m" represents the letter "T" followed by any single character, followed by the letter "m," and would retrieve rows that include the words Tim, Tom, or T3m.

ACCESS USER NOTE

Access uses different wildcard symbols. The symbol for any collection of characters is the asterisk (*), as shown in Figure 4-18. The symbol for an individual character is the question mark (?).

```
SELECT CUSTOMER_NUM, CUSTOMER_NAME, STREET, CITY, STATE, ZIP
FROM CUSTOMER
WHERE STREET LIKE '*Central*';
```

FIGURE 4-18 Access SELECT command with wildcards

NOTE

In a large database, you should use wildcards only when absolutely necessary. Searches involving wildcards can be extremely slow to process.

Using the IN Operator

An **IN clause**, which consists of the IN operator followed by a collection of values, provides a concise way of phrasing certain conditions, as Example 13 illustrates. You will see another use for the IN clause in more complex examples later in this chapter.

EXAMPLE 13

List the number, name, and credit limit for each customer with a credit limit of $5,000, $10,000, or $15,000.

In this query, you can use an IN clause to determine whether a credit limit is $5,000, $10,000, or $15,000. You could obtain the same answer by using the condition WHERE

CREDIT_LIMIT = 5000 OR CREDIT_LIMIT = 10000 OR CREDIT_LIMIT = 15000. The approach shown in Figure 4-19 is simpler because the IN clause contains a collection of values: 5000, 10000, and 15000. The condition is true for those rows in which the value in the CREDIT_LIMIT column is in this collection.

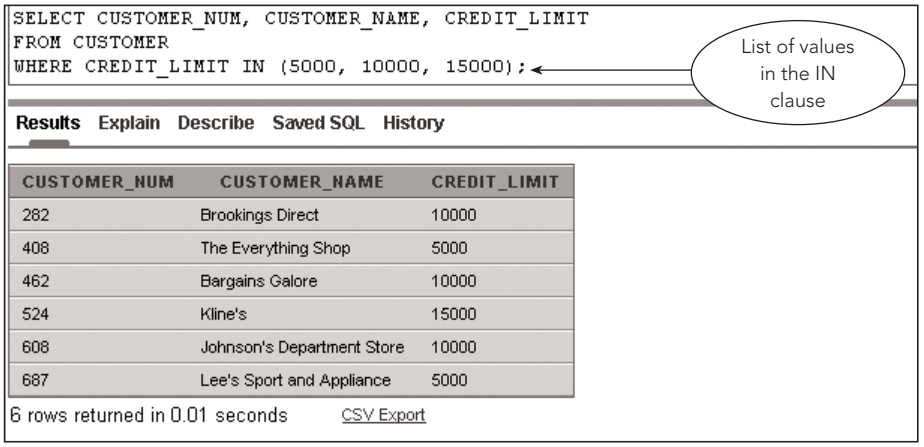


FIGURE 4-19 SELECT command with an IN clause

SORTING

Recall that the order of rows in a table is immaterial to the DBMS. From a practical standpoint, this means that when you query a relational database, there is no defined order in which to display the results. Rows might be displayed in the order in which the data was originally entered, but even this is not certain. If the order in which the data is displayed is important, you can specifically request that the results appear in a desired order. In SQL, you specify the results order by using the ORDER BY clause.

Using the ORDER BY Clause

You use the **ORDER BY clause** to list data in a specific order, as shown in Example 14.

EXAMPLE 14

List the number, name, and balance of each customer. Order (sort) the output in ascending (increasing) order by balance.

The column on which to sort data is called a **sort key** or simply a **key**. In Example 14, you need to order the output by balance, so the sort key is the BALANCE column. To sort the output, use an ORDER BY clause followed by the sort key. If you do not specify a sort order, the default is ascending. The query appears in Figure 4-20.

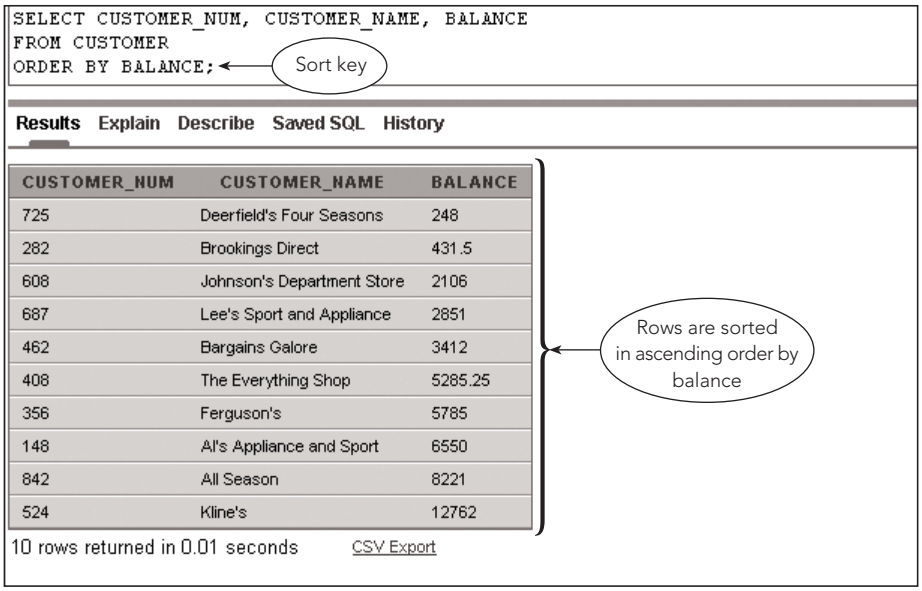


FIGURE 4-20 SELECT command to sort rows

Additional Sorting Options

Sometimes you might need to sort data using more than one key, as shown in Example 15.

EXAMPLE 15

List the number, name, and credit limit of each customer. Order the customers by name within descending credit limit. (In other words, first sort the customers by credit limit in descending order. Within each group of customers that have a common credit limit, sort the customers by name in ascending order.)

Example 15 involves two new ideas: sorting on multiple keys—CREDIT_LIMIT and CUSTOMER_NAME—and sorting one of the keys in descending order. When you need to sort data on two columns, the more important column (in this case, CREDIT_LIMIT) is called the **major sort key** (or the **primary sort key**) and the less important column (in this case, CUSTOMER_NAME) is called the **minor sort key** (or the **secondary sort key**). To sort on multiple keys, you list the keys in order of importance in the ORDER BY clause. To sort in descending order, you follow the name of the sort key with the **DESC** operator, as shown in Figure 4-21.

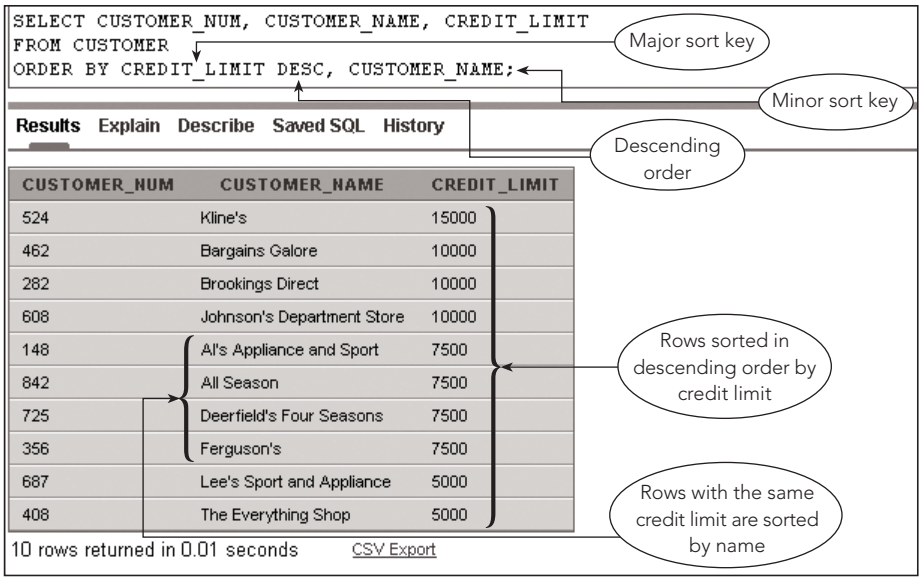


FIGURE 4-21 SELECT command to sort data using multiple sort keys

USING FUNCTIONS

SQL uses special functions, called **aggregate functions**, to calculate sums, averages, counts, maximum values, and minimum values. These functions apply to *groups* of rows. They could apply to all the rows in a table (for example, calculating the average balance of all customers). They also could apply to those rows satisfying some particular condition (for example, the average balance of all customers of sales rep 20). The descriptions of the aggregate functions appear in Figure 4-22.

Function	Description
AVG	Calculates the average value in a column
COUNT	Determines the number of rows in a table
MAX	Determines the maximum value in a column
MIN	Determines the minimum value in a column
SUM	Calculates a total of the values in a column

FIGURE 4-22 SQL aggregate functions

Using the COUNT Function

The **COUNT** function, as illustrated in Example 16, counts the number of rows in a table.

EXAMPLE 16

How many parts are in item class HW?

For this query, you need to determine the total number of rows in the PART table with the value HW in the CLASS column. You could count the part numbers in the query results, or the number of part descriptions, or the number of entries in any other column. It doesn't matter which column you choose because all columns should provide the same answer. Rather than arbitrarily selecting one column, most SQL implementations let you use the asterisk (*) to represent any column, as shown in Figure 4-23.

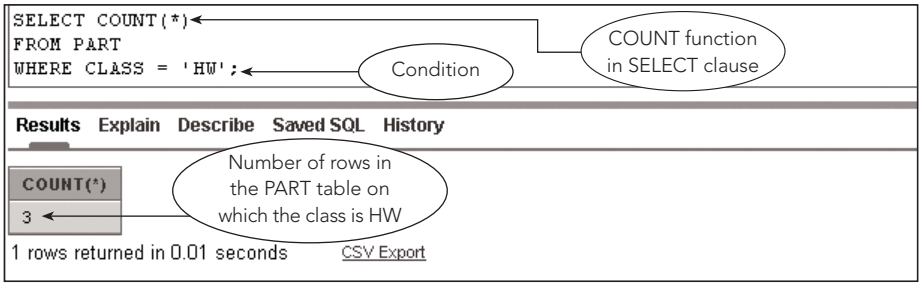


FIGURE 4-23 SELECT command to count rows

You also can count the number of rows in a query by selecting a specific column instead of using the asterisk, as follows:

```
SELECT COUNT(PART_NUM)
FROM PART
WHERE CLASS = 'HW';
```

Using the SUM Function

If you need to calculate the total of all customers' balances, you can use the **SUM** function, as illustrated in Example 17.

EXAMPLE 17

Find the total number of Premiere Products customers and the total of their balances.

When you use the SUM function, you must specify the column to total, and the column's data type must be numeric. (How could you calculate a sum of names or addresses?) Figure 4-24 shows the query.

```
SELECT COUNT(*), SUM(BALANCE)
FROM CUSTOMER;
```

Results Explain Describe Saved SQL History

COUNT(*)	SUM(BALANCE)
10	47651.75

1 rows returned in 0.01 seconds CSV Export

FIGURE 4-24 SELECT command to count rows and calculate a total

Using the AVG, MAX, and MIN Functions

Using the AVG, MAX, and MIN functions is similar to using SUM, except that different statistics are calculated. **AVG** calculates the average value in a numeric range, **MAX** calculates the maximum value in a numeric range, and **MIN** calculates the minimum value in a numeric range.

EXAMPLE 18

Find the sum of all balances, the average balance, the maximum balance, and the minimum balance of all Premiere Products customers.

Figure 4-25 shows the query and the results.

```
SELECT SUM(BALANCE), AVG(BALANCE), MAX(BALANCE), MIN(BALANCE)
FROM CUSTOMER;
```

Results Explain Describe Saved SQL History

SUM(BALANCE)	AVG(BALANCE)	MAX(BALANCE)	MIN(BALANCE)
47651.75	4765.175	12762	248

1 rows returned in 0.01 seconds CSV Export

FIGURE 4-25 SELECT command with several functions

NOTE

When you use the SUM, AVG, MAX, or MIN functions, SQL ignores any null value(s) in the column and eliminates them from the computations.

Null values in numeric columns can produce strange results when statistics are computed. Suppose the BALANCE column accepts null values, there are currently four customers in the CUSTOMER table, and their respective balances are $100, $200, $300, and null (unknown). When you calculate the average balance, SQL ignores the null value and obtains a result of $200 (($100 + $200 + $300) / 3). Similarly, when you calculate the total of the balances, SQL ignores the null value and calculates a total of $600. When you count the number of customers in the table, however, SQL includes the row containing the null value, and the result is 4. Thus the total of the balances ($600) divided by the number of customers (4) results in an average balance of $150!

NOTE

You can use an AS clause with a function. For example, the following command computes a sum of the BALANCE column and displays the column heading as TOTAL_BALANCE in the query results:

```
SELECT SUM(BALANCE) AS TOTAL_BALANCE
FROM CUSTOMER;
```

Using the DISTINCT Operator

In some situations, the **DISTINCT** operator is useful when used in conjunction with the COUNT function because it eliminates duplicate values in the query results. Examples 19 and 20 illustrate the most common uses of the DISTINCT operator.

EXAMPLE 19

Find the number of each customer that currently has an open order (that is, an order currently in the ORDERS table).

The command seems fairly simple. When a customer currently has an open order, there must be at least one row in the ORDERS table on which that customer's number appears. You could use the query shown in Figure 4-26 to find the customer numbers with open orders.

```
SELECT CUSTOMER_NUM
FROM ORDERS;
```

Results Explain Describe Saved SQL History

CUSTOMER_NUM
148
356
408
282
608
148
608

Customer 608 has two orders on file

7 rows returned in 0.04 seconds CSV Export

FIGURE 4-26 Numbers of customers with open orders

Notice that customer numbers 148 and 608 each appear more than once in the results; this means that both customers currently have more than one open order in the ORDERS table. Suppose you want to list each customer only once, as illustrated in Example 20.

EXAMPLE 20

Find the number of each customer that currently has an open order. List each customer only once.

To ensure uniqueness, you can use the DISTINCT operator, as shown in Figure 4-27.

```
SELECT DISTINCT(CUSTOMER_NUM)
FROM ORDERS;
```

Results Explain Describe Saved SQL History

CUSTOMER_NUM
282
148
608
356
408

5 rows returned in 0.04 seconds CSV Export

FIGURE 4-27 Numbers of customers with open orders and with duplicates removed

You might wonder about the relationship between COUNT and DISTINCT, because both involve counting rows. Example 21 identifies the differences.

EXAMPLE 21

Count the number of customers that currently have open orders.

The query shown in Figure 4-28 counts the number of customers using the CUSTOMER_NUM column.

```
SELECT COUNT(CUSTOMER_NUM)
FROM ORDERS;
```

Results Explain Describe Saved SQL History

COUNT(CUSTOMER_NUM)
7

1 rows returned in 0.01 seconds CSV Export

FIGURE 4-28 Count that includes duplicate customer numbers

Q & A

Question: What is wrong with the query results shown in Figure 4-28?
Answer: The answer, 7, is the result of counting the customers that have open orders multiple times—once for each separate order currently on file. The result counts each customer number and does not eliminate duplicate customer numbers to provide an accurate count of the number of customers.

Some SQL implementations, including Oracle and SQL Server (but not Access), allow you to use the DISTINCT operator to calculate the correct count, as shown in Figure 4-29.

```
SELECT COUNT(DISTINCT(CUSTOMER_NUM))
FROM ORDERS;
```

Results Explain Describe Saved SQL History

COUNT(DISTINCT(CUSTOMER_NUM))
5

1 rows returned in 0.04 seconds CSV Export

FIGURE 4-29 Count that excludes duplicate customer numbers (using DISTINCT within COUNT)

NESTING QUERIES

Sometimes obtaining the results you need requires two or more steps, as shown in the next two examples.

EXAMPLE 22

List the number of each part in class AP.

The command to obtain the answer is shown in Figure 4-30.

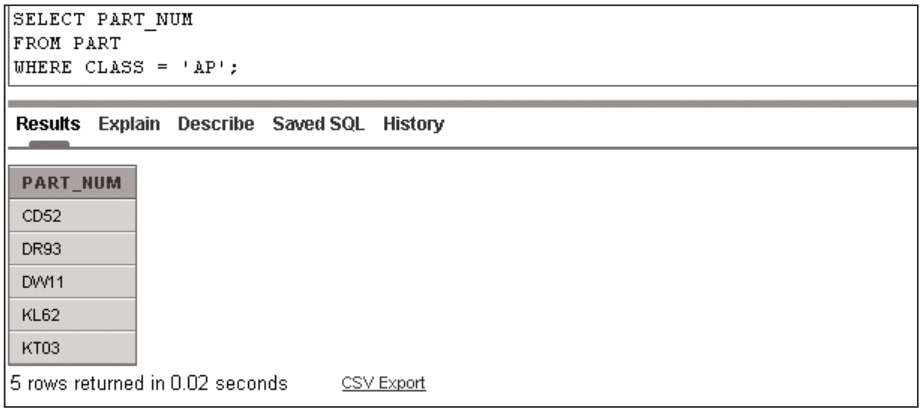


FIGURE 4-30 Selecting all parts in class AP

EXAMPLE 23

List the order numbers that contain an order line for a part in class AP.

Example 23 asks you to find the order numbers in the ORDER_LINE table that correspond to the part numbers in the results of the query used in Example 22. After viewing those results (CD52, DR93, DW11, KL62, and KT03), you can use the command shown in Figure 4-31.

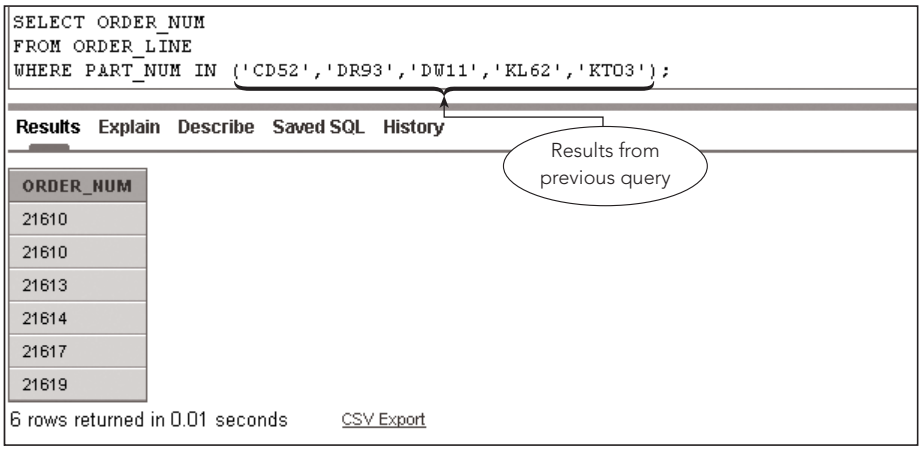


FIGURE 4-31 Query using the results from Figure 4-30

Subqueries

It is possible to place one query inside another. The inner query is called a **subquery**. The subquery is evaluated first. After the subquery has been evaluated, the outer query can use the results of the subquery to find its results, as shown in Example 24.

EXAMPLE 24

Find the answer to Examples 22 and 23 in one step.

You can find the same result as in the previous two examples in a single step by using a subquery. In Figure 4-32, the command shown in parentheses is the subquery. This subquery is evaluated first, producing a temporary table. The temporary table is used only to evaluate the query—it is not available to the user or displayed—and it is deleted after the evaluation of the query is complete. In this example, the temporary table has only a single column (PART_NUM) and five rows (CD52, DR93, DW11, KL62, and KT03). The outer query is evaluated next. In this case, the outer query retrieves the order number on every row in the ORDER_LINE table for which the part number is in the results of the subquery. Because that table contains only the part numbers in class AP, the results display the desired list of order numbers.

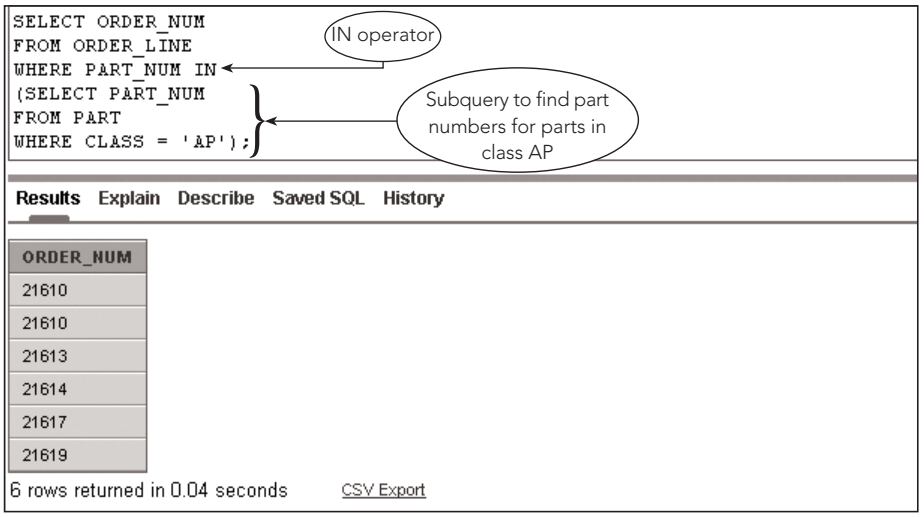


FIGURE 4-32 Using the IN operator and a subquery

Figure 4-32 shows duplicate order numbers in the results. To eliminate this duplication, you can use the DISTINCT operator as follows:

```
SELECT DISTINCT(ORDER_NUM)
FROM ORDER_LINE
WHERE PART_NUM IN
(SELECT PART_NUM
FROM PART
WHERE CLASS = 'AP');
```

The results of this query will display each order number only once.

EXAMPLE 25

List the number, name, and balance for each customer whose balance exceeds the average balance of all customers.

In this case, you use a subquery to obtain the average balance. Because this subquery produces a single number, you can compare each customer's balance with this number, as shown in Figure 4-33.

```
SELECT CUSTOMER_NUM, CUSTOMER_NAME, BALANCE
FROM CUSTOMER
WHERE BALANCE >
(SELECT AVG(BALANCE)
FROM CUSTOMER);
```

Results Explain Describe Saved SQL History

CUSTOMER_NUM	CUSTOMER_NAME	BALANCE
148	Al's Appliance and Sport	6550
356	Ferguson's	5785
408	The Everything Shop	5285.25
524	Kline's	12762
842	All Season	8221

5 rows returned in 0.01 seconds CSV Export

FIGURE 4-33 Query using an operator and a subquery

NOTE

You cannot use the condition BALANCE > AVG(BALANCE) in the WHERE clause; you must use a subquery to obtain the average balance. Then you can use the results of the subquery in a condition, as illustrated in Figure 4-33.

GROUPING

Grouping creates groups of rows that share some common characteristic. If you group customers by credit limit, for example, the first group contains customers with $5,000 credit limits, the second group contains customers with $7,500 credit limits, and so on. If, on the other hand, you group customers by sales rep number, the first group contains those customers represented by sales rep number 20, the second group contains those customers represented by sales rep number 35, and the third group contains those customers represented by sales rep number 65.

When you group rows, any calculations indicated in the SELECT command are performed for the entire group. For example, if you group customers by rep number and the query requests the average balance, the results include the average balance for the group of customers represented by rep number 20, the average balance for the group represented by rep number 35, and the average balance for the group represented by rep number 65. The following examples illustrate this process.

Using the GROUP BY Clause

The **GROUP BY clause** lets you group data on a particular column, such as REP_NUM, and then calculate statistics, when desired, as shown in Example 26.

EXAMPLE 26

For each sales rep, list the rep number and the average balance of the rep's customers.

Because you need to group customers by rep number and then calculate the average balance for all customers in each group, you must use the GROUP BY clause. In this case, GROUP BY REP_NUM puts customers with the same rep number into separate groups. Any statistics indicated in the SELECT command are calculated for each group. It is important to note that the GROUP BY clause does not sort the data in a particular order; you must use the ORDER BY clause to sort data. Assuming that the results should be ordered by rep number, you can use the command shown in Figure 4-34.

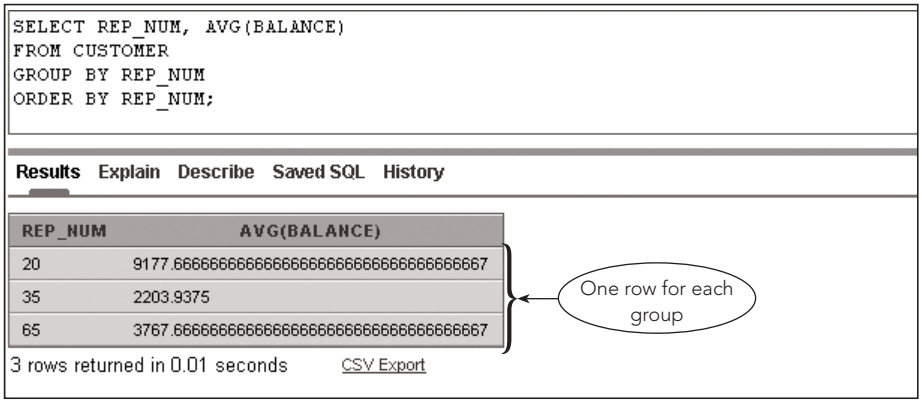


FIGURE 4-34 Grouping records on a column

When rows are grouped, one line of output is produced for each group. The only things that can be displayed are statistics calculated for the group or columns whose values are the same for all rows in a group.

Q & A

Question: Is it appropriate to display the rep number in the query for Example 26?
Answer: Yes, because the rep number in one row in a group must be the same as the rep number in any other row in the group.

Q & A

Question: Would it be appropriate to display a customer number in the query for Example 26?
Answer: No, because the customer number varies on the rows in a group. (The same rep is associated with many customers.) The DBMS would not be able to determine which customer number to display for the group, and would display an error message if you attempt to display a customer number.

Using a HAVING Clause

The HAVING clause is used to restrict the groups that are included, as shown in Example 27.

EXAMPLE 27

Repeat the previous example, but list only those reps who represent fewer than four customers.

The only difference between Examples 26 and 27 is the restriction to display only those reps who represent fewer than four customers. This restriction does not apply to individual rows but rather to *groups*. Because the WHERE clause applies only to rows, you cannot use it to accomplish the kind of selection that is required. Fortunately, the HAVING clause does for groups what the WHERE clause does for rows. The **HAVING clause** limits the groups that are included in the results. In Figure 4-35, the row created for a group is displayed only when the count of the number of rows in the group is less than four; in addition, all groups are ordered by rep number.

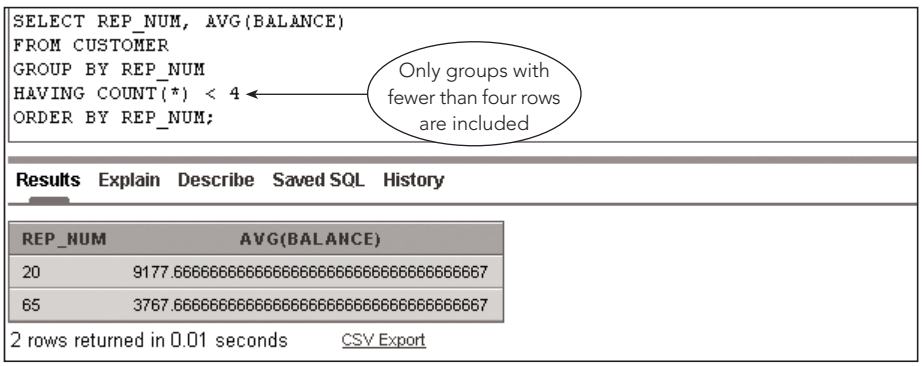


FIGURE 4-35 Restricting the groups to include in the results

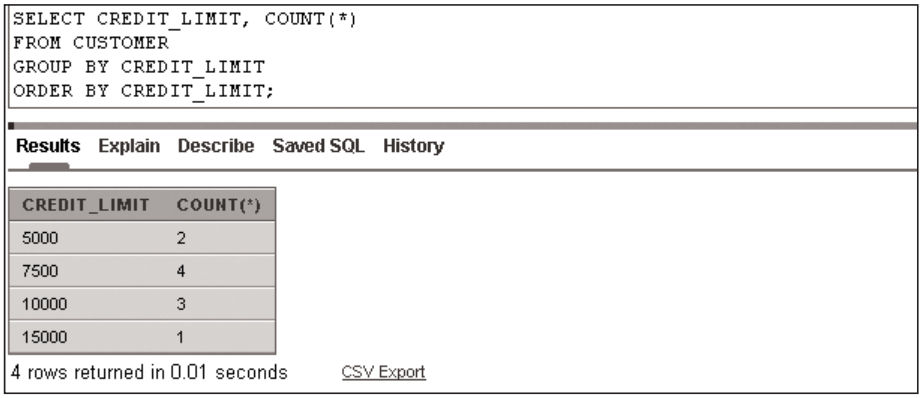
HAVING vs. WHERE

Just as you can use the WHERE clause to limit the *rows* that are included in a query's result, you can use the HAVING clause to limit the *groups* that are included. The following examples illustrate the difference between these two clauses.

EXAMPLE 28

List each credit limit and the number of customers having each credit limit.

To count the number of customers that have a given credit limit, you must group the data by credit limit, as shown in Figure 4-36.



```
SELECT CREDIT_LIMIT, COUNT(*)
FROM CUSTOMER
GROUP BY CREDIT_LIMIT
ORDER BY CREDIT_LIMIT;
```

Results Explain Describe Saved SQL History

CREDIT_LIMIT	COUNT(*)
5000	2
7500	4
10000	3
15000	1

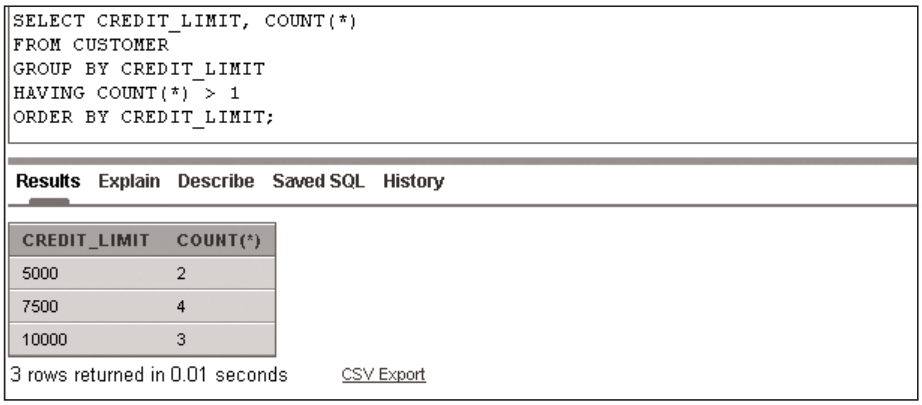
4 rows returned in 0.01 seconds CSV Export

FIGURE 4-36 Counting the number of rows in each group

EXAMPLE 29

Repeat Example 28, but list only those credit limits held by more than one customer.

Because this condition involves a group total, the query includes a HAVING clause, as shown in Figure 4-37.



```
SELECT CREDIT_LIMIT, COUNT(*)
FROM CUSTOMER
GROUP BY CREDIT_LIMIT
HAVING COUNT(*) > 1
ORDER BY CREDIT_LIMIT;
```

Results Explain Describe Saved SQL History

CREDIT_LIMIT	COUNT(*)
5000	2
7500	4
10000	3

3 rows returned in 0.01 seconds CSV Export

FIGURE 4-37 Displaying groups that contain more than one row

EXAMPLE 30

List each credit limit and the number of customers of sales rep 20 that have this limit.

The condition involves only rows, so using the WHERE clause is appropriate, as shown in Figure 4-38.

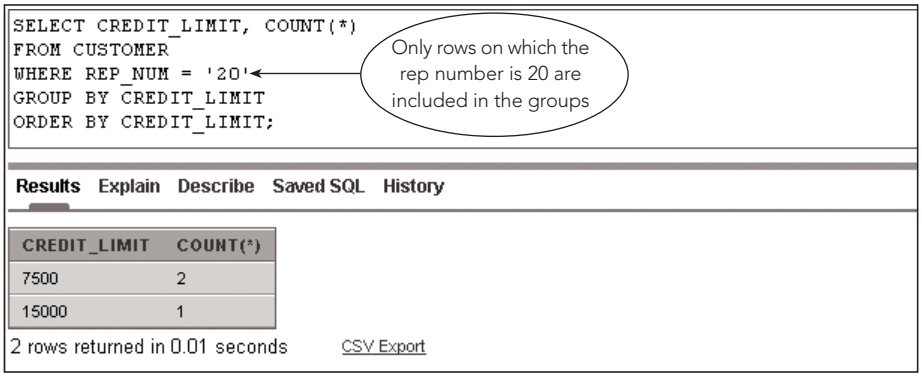


FIGURE 4-38 Restricting the rows to be grouped

EXAMPLE 31

Repeat Example 30, but list only those credit limits held by more than one customer.

Because the conditions involve rows and groups, you must use both a WHERE clause and a HAVING clause, as shown in Figure 4-39.

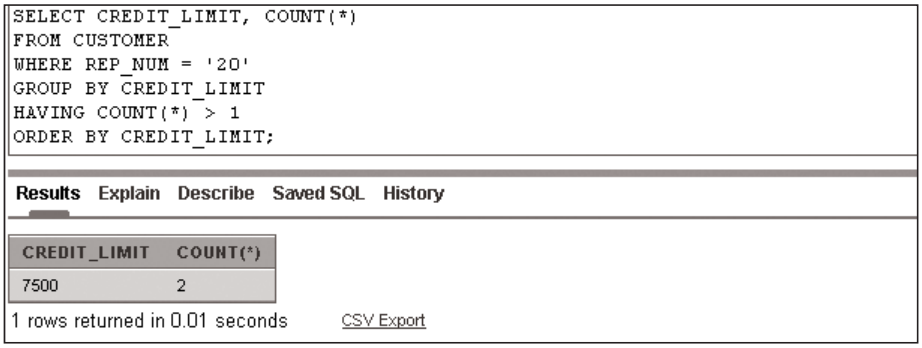


FIGURE 4-39 Restricting the rows and the groups

In Example 31, rows from the original table are evaluated only when the sales rep number is 20. These rows then are grouped by credit limit and the count is calculated. Only groups for which the calculated count is greater than one are displayed.

NULLS

Sometimes a condition involves a column that can accept null values, as illustrated in Example 32.

EXAMPLE 32

List the number and name of each customer with a null (unknown) street value.

You might expect the condition to be something like STREET = NULL. The correct format actually uses the **IS NULL** operator (STREET IS NULL), as shown in Figure 4-40. (To select a customer whose street is not null, use the **IS NOT NULL** operator (STREET IS NOT NULL).) In the current Premiere Products database, no customer has a null street value; therefore, no rows are retrieved in the query results.

```
SELECT CUSTOMER_NUM, CUSTOMER_NAME
FROM CUSTOMER
WHERE STREET IS NULL;
```

Results Explain Describe Saved SQL History

no data found

FIGURE 4-40 Selecting rows containing null values in the STREET column

SUMMARY OF SQL CLAUSES, FUNCTIONS, AND OPERATORS

In this chapter, you learned how to create queries that retrieve data from a single table by constructing appropriate SELECT commands. In the next chapter, you will learn how to create queries that retrieve data from multiple tables. The queries you created in this chapter used the clauses, functions, and operators shown in Figure 4-41.

Clause, function, or operator	Description
AND operator	Specifies that all simple conditions must be true for the compound condition to be true
AVG function	Calculates the average value in a numeric range
BETWEEN operator	Specifies a range of values in a condition
COUNT function	Counts the number of rows in a table
DESC operator	Sorts the query results in descending order based on the column name
DISTINCT operator	Ensures uniqueness in the condition by eliminating redundant values
FROM clause	Indicates the table from which to retrieve the specified columns
GROUP BY clause	Groups rows based on the specified column
HAVING clause	Limits a condition to the groups that are included
IN clause	Uses the IN operator to find a value in a group of values specified in the condition
IS NOT NULL operator	Finds rows that do not contain a null value in the specified column
IS NULL operator	Finds rows that contain a null value in the specified column
LIKE operator	Indicates a pattern of characters to find in a condition
MAX function	Calculates the maximum value in a numeric range
MIN function	Calculates the minimum value in a numeric range
NOT operator	Reverses the truth or falsity of the original condition
OR operator	Specifies that the compound condition is true whenever any of the simple conditions is true
ORDER BY clause	Lists the query results in the specified order based on the column name
SELECT clause	Specifies the columns to retrieve in the query
SUM function	Totals the values in a numeric range
WHERE clause	Specifies any conditions for the query

FIGURE 4-41 SQL query clauses, functions, and operators

Chapter Summary

- The basic form of the SQL SELECT command is SELECT-FROM-WHERE. Specify the columns to be listed after the word SELECT (or type an asterisk (*) to select all columns), and then specify the table name that contains these columns after the word FROM. Optionally, you can include one or more conditions after the word WHERE.
- Simple conditions are written in the following form: column name, comparison operator, column name or value. Simple conditions can involve any of the comparison operators: =, >, >=, <, <=, < >, or !=.
- You can form compound conditions by combining simple conditions using the AND, OR, and NOT operators.
- Use the BETWEEN operator to indicate a range of values in a condition.
- Use computed columns in SQL commands by using arithmetic operators and writing the computation in place of a column name. You can assign a name to the computed column by following the computation with the word AS and then the desired name.
- To check for a value in a character column that is similar to a particular string of characters, use the LIKE operator. In Oracle and SQL Server, the percent (%) wildcard represents any collection of characters, and the underscore (_) wildcard represents any single character. In Access, the asterisk (*) wildcard represents any collection of characters, and the question mark (?) wildcard represents any single character.
- To determine whether a column contains a value in a set of values, use the IN operator.
- Use an ORDER BY clause to sort data. List sort keys in order of importance. To sort in descending order, follow the sort key with the DESC operator.
- SQL processes the aggregate functions COUNT, SUM, AVG, MAX, and MIN. These calculations apply to groups of rows.
- To avoid duplicates in a query that uses an aggregate function, precede the column name with the DISTINCT operator.
- When one SQL query is placed inside another, it is called a subquery. The inner query (the subquery) is evaluated first.
- Use a GROUP BY clause to group data.
- Use a HAVING clause to restrict the output to certain groups.
- Use the IS NULL operator in a WHERE clause to find rows containing a null value in a particular column. Use the IS NOT NULL operator in a WHERE clause to find rows that do not contain a null value.

Key Terms

aggregate function
AND
AND condition
AVG
BETWEEN
compound condition
computed column
COUNT
DESC
DISTINCT

FROM clause
GROUP BY clause
grouping
HAVING clause
IN clause
IS NOT NULL
IS NULL
key
LIKE
major sort key
MAX
MIN
minor sort key
NOT
NOT condition
OR
OR condition
ORDER BY clause
primary sort key
query
secondary sort key
SELECT clause
simple condition
sort key
subquery
SUM
WHERE clause

Review Questions

1. Describe the basic form of the SQL SELECT command.
2. How do you form a simple condition?
3. How do you form a compound condition?
4. In SQL, what operator do you use to determine whether a value is between two other values without using an AND condition?
5. How do you use a computed column in SQL? How do you name the computed column?
6. In which clause would you use a wildcard in a condition?
7. What wildcards are available in Oracle, and what do they represent?
8. How do you determine whether a column contains one of a particular set of values without using an AND condition?
9. How do you sort data?
10. How do you sort data on more than one sort key? What is the more important key called? What is the less important key called?
11. How do you sort data in descending order?
12. What are the SQL aggregate functions?
13. How do you avoid including duplicate values in a query's results?
14. What is a subquery?
15. How do you group data in an SQL query?
16. When grouping data in a query, how do you restrict the output to only those groups satisfying some condition?
17. How do you find rows in which a particular column contains a null value?

18. Use your favorite Web browser and Web search engine to find out how to enter a date in an SQL query in Oracle, Access, and SQL Server. Using the information you find, complete the following SQL command for each of the three DBMSs (Oracle, Access, and SQL Server) to list orders placed on October 20, 2010:

```
SELECT *
FROM ORDERS
WHERE ORDER_DATE =
```

Be sure to reference the URLs that contain the information.

Exercises

Premiere Products

Use SQL and the Premiere Products database (see Figure 1-2 in Chapter 1) to complete the following exercises. If directed to do so by your instructor, use the information provided with the Chapter 3 Exercises to print your output.

1. List the part number, description, and price for all parts.
2. List all rows and columns for the complete ORDERS table.
3. List the names of customers with credit limits of $10,000 or more.
4. List the order number for each order placed by customer number 608 on 10/23/2010. (*Hint:* If you need help, use the discussion of the DATE data type in Figure 3-11 in Chapter 3.)
5. List the number and name of each customer represented by sales rep 35 or sales rep 65.
6. List the part number and part description of each part that is not in item class AP.
7. List the part number, description, and number of units on hand for each part that has between 10 and 25 units on hand, including both 10 and 25. Do this two ways.
8. List the part number, part description, and on-hand value (units on hand * unit price) of each part in item class SG. (On-hand value is really units on hand * cost, but there is no COST column in the PART table.) Assign the name ON_HAND_VALUE to the computed column.
9. List the part number, part description, and on-hand value for each part whose on-hand value is at least $7,500. Assign the name ON_HAND_VALUE to the computed column.
10. Use the IN operator to list the part number and part description of each part in item class AP or SG.
11. Find the number and name of each customer whose name begins with the letter “B.”
12. List all details about all parts. Order the output by part description.
13. List all details about all parts. Order the output by part number within warehouse. (That is, order the output by warehouse and then by part number.)
14. How many customers have balances that are more than their credit limits?
15. Find the total of the balances for all customers represented by sales rep 65 with balances that are less than their credit limits.
16. List the part number, part description, and on-hand value of each part whose number of units on hand is more than the average number of units on hand for all parts. (*Hint:* Use a subquery.)

17. What is the price of the least expensive part in the database?
18. What is the part number, description, and price of the least expensive part in the database? (*Hint:* Use a subquery.)
19. List the sum of the balances of all customers for each sales rep. Order and group the results by sales rep number.
20. List the sum of the balances of all customers for each sales rep, but restrict the output to those sales reps for which the sum is more than $10,000.
21. List the part number of any part with an unknown description.

Henry Books

Use SQL and the Henry Books database (Figures 1-4 through 1-7 in Chapter 1) to complete the following exercises. If directed to do so by your instructor, use the information provided with the Chapter 3 Exercises to print your output.

1. List the book code and book title of each book.
2. List the complete BRANCH table.
3. List the name of each publisher located in Boston.
4. List the name of each publisher not located in New York.
5. List the name of each branch that has at least nine employees.
6. List the book code and book title of each book that has the type HOR.
7. List the book code and book title of each book that has the type HOR and is in paperback.
8. List the book code and book title of each book that has the type HOR or is published by the publisher with the publisher code SC.
9. List the book code, book title, and price of each book with a price between $15 and $25.
10. List the book code and book title of each book that has the type MYS and a price of less than $20.
11. Customers who are part of a special program get a 10 percent discount off regular book prices. List the book code, book title, and discounted price of each book. Use DISCOUNTED_PRICE as the name for the computed column, which should calculate 90 percent of the current price (100 percent less a 10 percent discount).
12. Find the name of each publisher containing the word "and." (*Hint:* Be sure that your query selects only those publishers that contain the word "and" and not those that contain the letters "and" in the middle of a word. For example, your query should select the publisher named "Farrar Straus and Giroux," but should *not* select the publisher named "Random House.")
13. List the book code and book title of each book that has the type SFI, MYS, or HOR. Use the IN operator in your command.
14. Repeat Exercise 13, but also list the books in alphabetical order by title.
15. Repeat Exercise 13, but also include the price, and list the books in descending order by price. Within a group of books having the same price, further order the books by title.
16. Display the list of book types in the database. List each book type only once.
17. How many books have the type SFI?

18. For each type of book, list the type and the average price.
19. Repeat Exercise 18, but consider only paperback books.
20. Repeat Exercise 18, but consider only paperback books for those types for which the average price is more than $10.
21. What are the title(s) and price(s) of the least expensive book(s) in the database?
22. What is the most expensive book in the database?
23. How many employees does Henry Books have?

Alexamara Marina Group

Use SQL and the Alexamara Marina Group database (Figures 1-8 through 1-12 in Chapter 1) to complete the following exercises. If directed to do so by your instructor, use the information provided with the Chapter 3 Exercises to print your output.

1. List the owner number, last name, and first name of every boat owner.
2. List the complete MARINA table (all rows and all columns).
3. List the last name and first name of every owner who lives in Rivard.
4. List the last name and first name of every owner who does not live in Rivard.
5. List the marina number and slip number for every slip whose length is equal to or less than 30 feet.
6. List the marina number and slip number for every boat with the type Ray 4025.
7. List the slip number for every boat with the type Ray 4025 that is located in marina 1.
8. List the boat name for each boat located in a slip whose length is between 25 and 30 feet.
9. List the slip number for every slip in marina 1 whose rental fee is less than $3,000.
10. Labor is billed at the rate of $60 per hour. List the slip ID, category number, estimated hours, and estimated labor cost for every service request. To obtain the estimated labor cost, multiply the estimated hours by 60. Use the column name ESTIMATED_COST for the estimated labor cost.
11. List the marina number and slip number for all slips containing a boat with the type Sprite 4000, Sprite 3000, or Ray 4025.
12. List the marina number, slip number, and boat name for all boats. Sort the results by boat name within marina number.
13. How many Dolphin 28 boats are stored at both marinas?
14. Calculate the total rental fees Alexamara receives each year based on the length of the slip.

This page contains answers for this chapter only.

CHAPTER 4—SINGLE-TABLE QUERIES

1. The basic form of the SELECT command is SELECT-FROM-WHERE. Specify the columns to be listed after the word SELECT (or type * to select all columns), and then specify the table name that contains these columns after the word FROM. Optionally, you can include condition(s) after the word WHERE.
3. You can form a compound condition by combining simple conditions and using the operators AND, OR, or NOT.
5. Use arithmetic operators and write the computation in place of a column name. You can assign a name to the computation by following the computation with the word AS and then the desired name.
7. In Oracle, the percent (%) wildcard represents any collection of characters. The underscore (_) wildcard represents any single character.
9. Use an ORDER BY clause.
11. To sort data in descending order, follow the sort key with the DESC operator.
13. To avoid duplicates, precede the column name with the DISTINCT operator.
15. Use a GROUP BY clause.
17. Use the IS NULL operator in the WHERE clause.

This page contains answers for this chapter only.