CHAPTER 5

MULTIPLE-TABLE QUERIES

LEARNING OBJECTIVES

Objectives

- Use joins to retrieve data from more than one table
- Use the IN and EXISTS operators to query multiple tables
- Use a subquery within a subquery
- Use an alias
- Join a table to itself
- Perform set operations (union, intersection, and difference)
- Use the ALL and ANY operators in a query
- Perform special operations (inner join, outer join, and product)

INTRODUCTION

In this chapter, you will learn how to use SQL to retrieve data from two or more tables using one SQL command. You will join tables together and examine how to obtain similar results using the SQL IN and EXISTS operators. Then you will use aliases to simplify queries and join a table to itself. You also will implement the set operations of union, intersection, and difference using SQL commands. You will examine two related SQL operators: ALL and ANY. Finally, you will perform inner joins, outer joins, and products.

QUERYING MULTIPLE TABLES

In Chapter 4, you learned how to retrieve data from a single table. Many queries require you to retrieve data from two or more tables. To retrieve data from multiple tables, you first must join the tables, and then formulate a query using the same commands that you use for single-table queries.

NOTE

In the following queries, your results might contain the same rows, but they might be listed in a different order. If order is important, you can include an ORDER BY clause in the query to ensure that the results are listed in the desired order.

Joining Two Tables

To retrieve data from more than one table, you must **join** the tables together by finding rows in the two tables that have identical values in matching columns. You can join tables by using a condition in the WHERE clause, as you will see in Example 1.

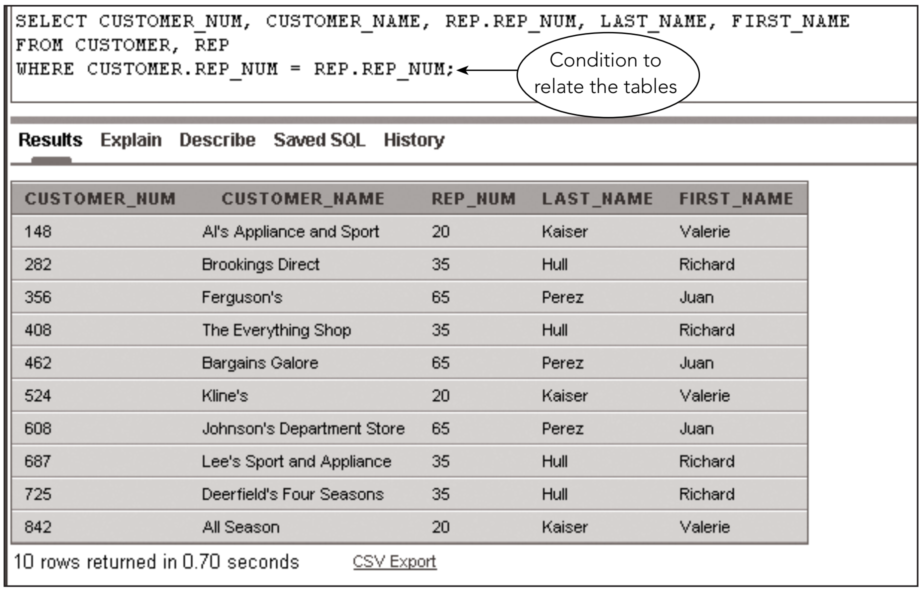
EXAMPLE 1

List the number and name of each customer, together with the number, last name, and first name of the sales rep who represents the customer.

Because the customer numbers and names are in the CUSTOMER table and the sales rep numbers and names are in the REP table, you need to include both tables in the SQL command so you can retrieve data from both tables. To join (relate) the tables, you construct the SQL command as follows:

1. In the SELECT clause, list all columns you want to display.
2. In the FROM clause, list all tables involved in the query.
3. In the WHERE clause, list the condition that restricts the data to be retrieved to only those rows from the two tables that match; that is, restrict it to the rows that have common values in matching columns.

As you learned in Chapter 2, it is often necessary to qualify a column name to specify the particular column you are referencing. Qualifying column names is especially important when joining tables because you must join tables on *matching* columns that frequently have identical column names. To qualify a column name, precede the name of the column with the name of the table, followed by a period. The matching columns in this example are both named REP_NUM—there is a column in the REP table named REP_NUM and a column in the CUSTOMER table that also is named REP_NUM. The REP_NUM column in the REP table is written as REP.REP_NUM and the REP_NUM column in the CUSTOMER table is written as CUSTOMER.REP_NUM. The query and its results appear in Figure 5-1.



```
SELECT CUSTOMER_NUM, CUSTOMER_NAME, REP.REP_NUM, LAST_NAME, FIRST_NAME
FROM CUSTOMER, REP
WHERE CUSTOMER.REP_NUM = REP.REP_NUM;
```

Condition to relate the tables
Results Explain Describe Saved SQL History
CSV Export

CUSTOMER_NUM	CUSTOMER_NAME	REP_NUM	LAST_NAME	FIRST_NAME
148	Al's Appliance and Sport	20	Kaiser	Valerie
282	Brookings Direct	35	Hull	Richard
356	Ferguson's	65	Perez	Juan
408	The Everything Shop	35	Hull	Richard
462	Bargains Galore	65	Perez	Juan
524	Kline's	20	Kaiser	Valerie
608	Johnson's Department Store	65	Perez	Juan
687	Lee's Sport and Appliance	35	Hull	Richard
725	Deerfield's Four Seasons	35	Hull	Richard
842	All Season	20	Kaiser	Valerie

```
10 rows returned in 0.70 seconds
```

FIGURE 5-1 Joining two tables with a single SQL command

When there is potential ambiguity in listing column names, you *must* qualify the columns involved in the query. It is permissible to qualify other columns as well, even when there is no possible confusion. Some people prefer to qualify all column names; in this text, however, you will qualify column names only when necessary.

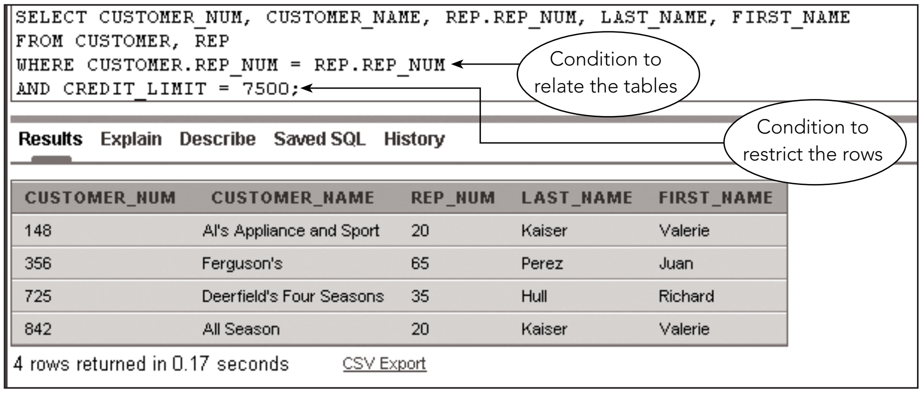
Q & A

Question: In the first row of output in Figure 5-1, the customer number is 148, and the customer name is Al's Appliance and Sport. These values represent the first row of the CUSTOMER table. Why is the sales rep number 20, the last name of the sales rep Kaiser, and the first name Valerie?
Answer: In the CUSTOMER table, the sales rep number for customer number 148 is 20. (This indicates that customer number 148 is *related* to sales rep number 20.) In the REP table, the last name of sales rep number 20 is Kaiser and the first name is Valerie.

EXAMPLE 2

List the number and name of each customer whose credit limit is $7,500, together with the number, last name, and first name of the sales rep who represents the customer.

In Example 1, you used a condition in the WHERE clause only to relate a customer with a sales rep to join the tables. Although relating a customer with a sales rep is essential in this example as well, you also need to restrict the output to only those customers whose credit limits are $7,500. You can restrict the rows by using a compound condition, as shown in Figure 5-2.



```
SELECT CUSTOMER_NUM, CUSTOMER_NAME, REP.REP_NUM, LAST_NAME, FIRST_NAME
FROM CUSTOMER, REP
WHERE CUSTOMER.REP_NUM = REP.REP_NUM
AND CREDIT_LIMIT = 7500;
```

Condition to relate the tables
Condition to restrict the rows
Results Explain Describe Saved SQL History
CSV Export

CUSTOMER_NUM	CUSTOMER_NAME	REP_NUM	LAST_NAME	FIRST_NAME
148	Al's Appliance and Sport	20	Kaiser	Valerie
356	Ferguson's	65	Perez	Juan
725	Deerfield's Four Seasons	35	Hull	Richard
842	All Season	20	Kaiser	Valerie

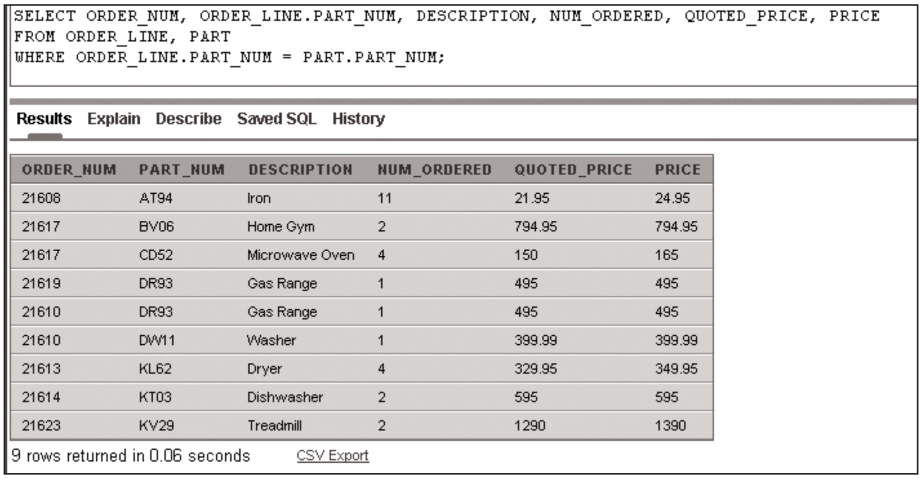
```
4 rows returned in 0.17 seconds
```

FIGURE 5-2 Restricting the rows in a join

EXAMPLE 3

For every part on order, list the order number, part number, part description, number of units ordered, quoted price, and unit price.

A part is considered "on order" when there is a row in the ORDER_LINE table in which the part appears. You can find the order number, number of units ordered, and quoted price in the ORDER_LINE table. To find the part description and the unit price, however, you need to look in the PART table. Then you need to find rows in the ORDER_LINE table and rows in the PART table that match (rows containing the same part number). The query and its results appear in Figure 5-3.



```
SELECT ORDER_NUM, ORDER_LINE.PART_NUM, DESCRIPTION, NUM_ORDERED, QUOTED_PRICE, PRICE
FROM ORDER_LINE, PART
WHERE ORDER_LINE.PART_NUM = PART.PART_NUM;
```

ORDER_NUM	PART_NUM	DESCRIPTION	NUM_ORDERED	QUOTED_PRICE	PRICE
21608	AT94	Iron	11	21.95	24.95
21617	BV06	Home Gym	2	794.95	794.95
21617	CD52	Microwave Oven	4	150	165
21619	DR93	Gas Range	1	495	495
21610	DR93	Gas Range	1	495	495
21610	DW11	Washer	1	399.99	399.99
21613	KL62	Dryer	4	329.95	349.95
21614	KT03	Dishwasher	2	595	595
21623	KV29	Treadmill	2	1290	1390

9 rows returned in 0.06 seconds CSV Export

FIGURE 5-3 Joining the ORDER_LINE and PART tables

Q & A

Question: Can you use PART.PART_NUM instead of ORDER_LINE.PART_NUM in the SELECT clause?
Answer: Yes. The values for these two columns match because they must satisfy the condition ORDER_LINE.PART_NUM = PART.PART_NUM.

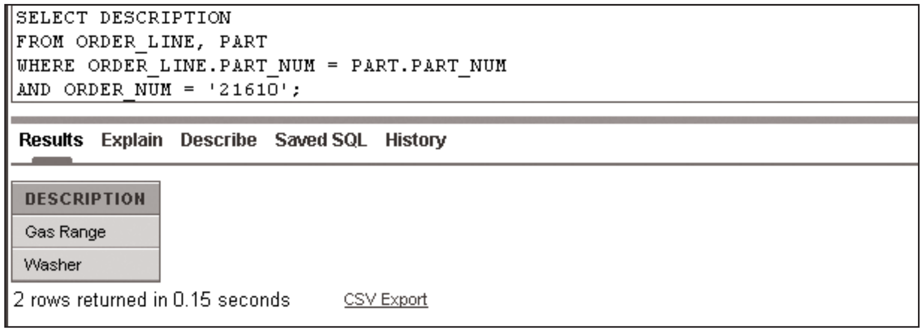
COMPARING JOINS, IN, AND EXISTS

You join tables in SQL by including a condition in the WHERE clause to ensure that matching columns contain equal values (for example, ORDER_LINE.PART_NUM = PART.PART_NUM). You can obtain similar results by using either the IN operator (described in Chapter 4) or the **EXISTS** operator with a subquery. The choice is a matter of personal preference because either approach obtains the same results. The following examples illustrate the use of each operator.

EXAMPLE 4

Find the description of each part included in order number 21610.

Because this query also involves retrieving data from the ORDER_LINE and PART tables (as illustrated in Example 3), you could approach it in a similar fashion. There are two basic differences, however, between Examples 3 and 4. First, the query in Example 4 does not require as many columns; second, it involves only order number 21610. Having fewer columns to retrieve means that there will be fewer columns listed in the SELECT clause. You can restrict the query to a single order by adding the condition ORDER_NUM = '21610' to the WHERE clause. The query and its results appear in Figure 5-4.



```
SELECT DESCRIPTION
FROM ORDER_LINE, PART
WHERE ORDER_LINE.PART_NUM = PART.PART_NUM
AND ORDER_NUM = '21610';
```

DESCRIPTION
Gas Range
Washer

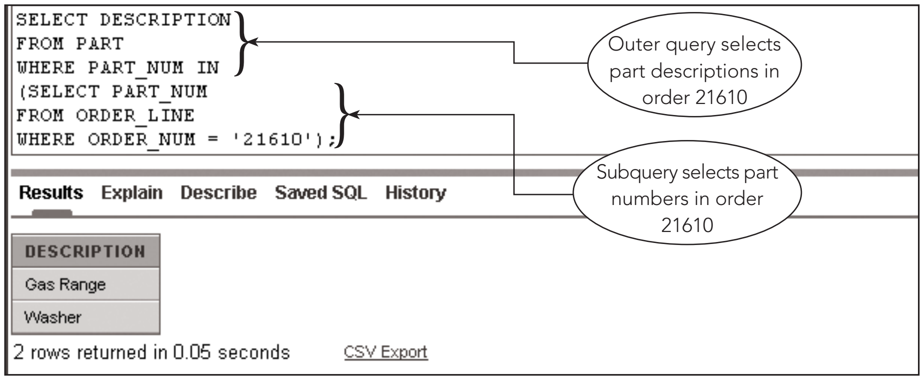
2 rows returned in 0.15 seconds

FIGURE 5-4 Restricting the rows when joining the ORDER_LINE and PART tables

Notice that the ORDER_LINE table is listed in the FROM clause, even though you do not need to display any columns from the ORDER_LINE table. The WHERE clause contains columns from the ORDER_LINE table, so it is necessary to include the table in the FROM clause.

Using the IN Operator

Another way to retrieve data from multiple tables in a query is to use the IN operator with a subquery. In Example 4, you first could use a subquery to find all part numbers in the ORDER_LINE table that appear in any row on which the order number is 21610. Then you could find the part description for any part whose part number is in this list. The query and its results appear in Figure 5-5.



```
SELECT DESCRIPTION
FROM PART
WHERE PART_NUM IN
(SELECT PART_NUM
FROM ORDER_LINE
WHERE ORDER_NUM = '21610');
```

Outer query selects part descriptions in order 21610
Subquery selects part numbers in order 21610
Results
Explain
Describe
Saved SQL
History
CSV Export

DESCRIPTION
Gas Range
Washer

2 rows returned in 0.05 seconds

FIGURE 5-5 Using the IN operator instead of a join to query two tables

In Figure 5-5, evaluating the subquery produces a temporary table consisting of those part numbers (DR93 and DW11) that are present in order number 21610. Executing the remaining portion of the query produces part descriptions for each part whose number is in this temporary table; in this case, Gas Range (DR93) and Washer (DW11).

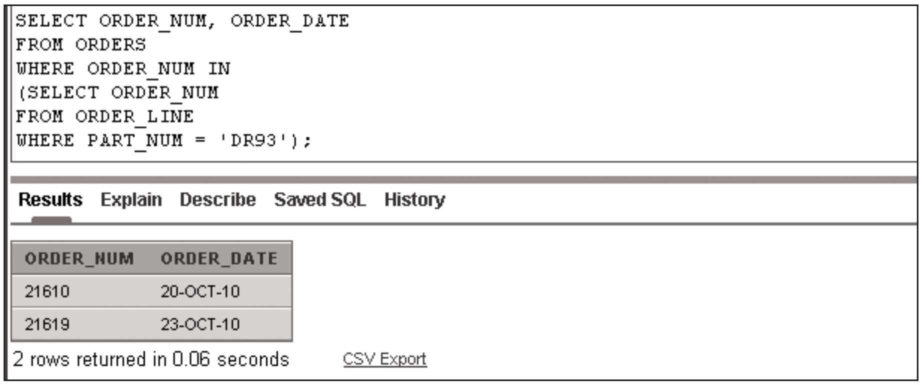
Using the EXISTS Operator

You also can use the EXISTS operator to retrieve data from more than one table, as shown in Example 5. The **EXISTS** operator checks for the existence of rows that satisfy some criterion.

E X A M P L E   5

Find the order number and order date for each order that contains part number DR93.

This query is similar to the one in Example 4, but this time the query involves the ORDERS table and not the PART table. In this case, you can write the query in either of the ways previously demonstrated. For example, you could use the IN operator with a subquery, as shown in Figure 5-6.



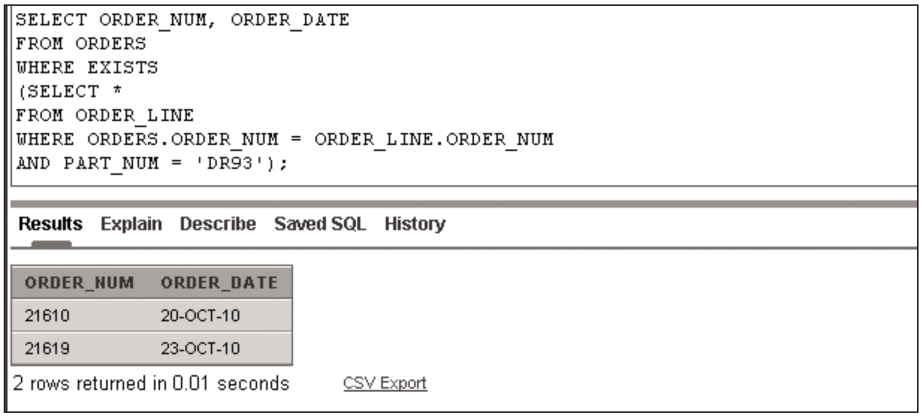
```
SELECT ORDER_NUM, ORDER_DATE
FROM ORDERS
WHERE ORDER_NUM IN
(SELECT ORDER_NUM
FROM ORDER_LINE
WHERE PART_NUM = 'DR93');
```

ORDER_NUM	ORDER_DATE
21610	20-OCT-10
21619	23-OCT-10

2 rows returned in 0.06 seconds

FIGURE 5-6 Using the IN operator to select order information

Using the EXISTS operator provides another approach to solving Example 5, as shown in Figure 5-7.



```
SELECT ORDER_NUM, ORDER_DATE
FROM ORDERS
WHERE EXISTS
(SELECT *
FROM ORDER_LINE
WHERE ORDERS.ORDER_NUM = ORDER_LINE.ORDER_NUM
AND PART_NUM = 'DR93');
```

```
ORDER_NUM   ORDER_DATE
21610       20-OCT-10
21619       23-OCT-10
2 rows returned in 0.01 seconds
```

FIGURE 5-7 Using the EXISTS operator to select order information

The subquery in Figure 5-7 is the first one you have seen that involves a table listed in the outer query. This type of subquery is called a **correlated subquery**. In this case, the ORDERS table, which is listed in the FROM clause of the outer query, is used in the subquery. For this reason, you need to qualify the ORDER_NUM column in the subquery (ORDERS.ORDER_NUM). You did not need to qualify the columns in the previous queries involving the IN operator.

The query shown in Figure 5-7 works as follows. For each row in the ORDERS table, the subquery is executed using the value of ORDERS.ORDER_NUM that occurs in that row. The inner query produces a list of all rows in the ORDER_LINE table in which ORDER_LINE.ORDER_NUM matches this value and in which PART_NUM is equal to DR93. You can precede a subquery with the EXISTS operator to create a condition that is true if one or more rows are obtained when the subquery is executed; otherwise, the condition is false.

To illustrate the process, consider order numbers 21610 and 21613 in the ORDERS table. Order number 21610 is included because a row exists in the ORDER_LINE table with this order number and part number DR93. When the subquery is executed, there will be at least one row in the results, which in turn makes the EXISTS condition true. Order number 21613, however, will not be included because no row exists in the ORDER_LINE table with this order number and part number DR93. There will be no rows contained in the results of the subquery, which in turn makes the EXISTS condition false.

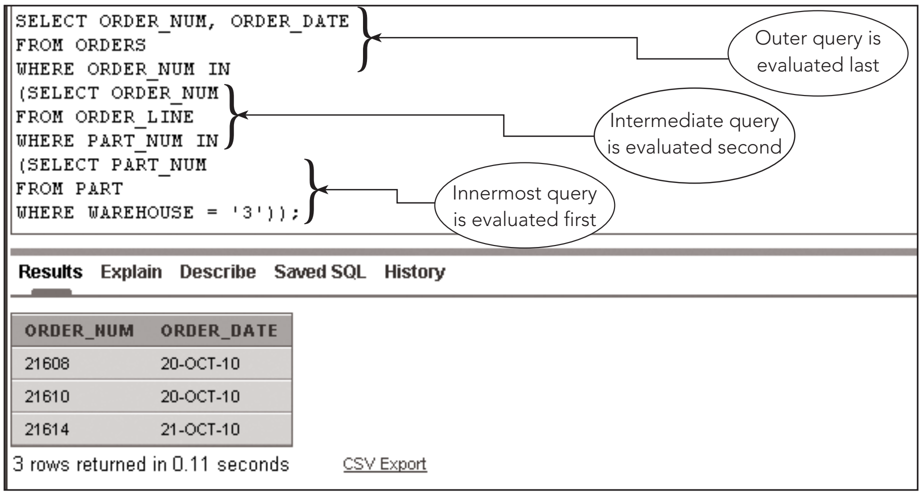
Using a Subquery Within a Subquery

You can use SQL to create a **nested subquery** (a subquery within a subquery), as illustrated in Example 6.

EXAMPLE 6

Find the order number and order date for each order that includes a part located in warehouse 3.

One way to approach this problem is first to determine the list of part numbers in the PART table for each part located in warehouse 3. Then you obtain a list of order numbers in the ORDER_LINE table with a corresponding part number in the part number list. Finally, you retrieve those order numbers and order dates in the ORDERS table for which the order number is in the list of order numbers obtained during the second step. The query and its results appear in Figure 5-8.



```
SELECT ORDER_NUM, ORDER_DATE
FROM ORDERS
WHERE ORDER_NUM IN
(SELECT ORDER_NUM
FROM ORDER_LINE
WHERE PART_NUM IN
(SELECT PART_NUM
FROM PART
WHERE WAREHOUSE = '3'));
```

Outer query is evaluated last
Intermediate query is evaluated second
Innermost query is evaluated first
Results Explain Describe Saved SQL History
CSV Export

ORDER_NUM	ORDER_DATE
21608	20-OCT-10
21610	20-OCT-10
21614	21-OCT-10

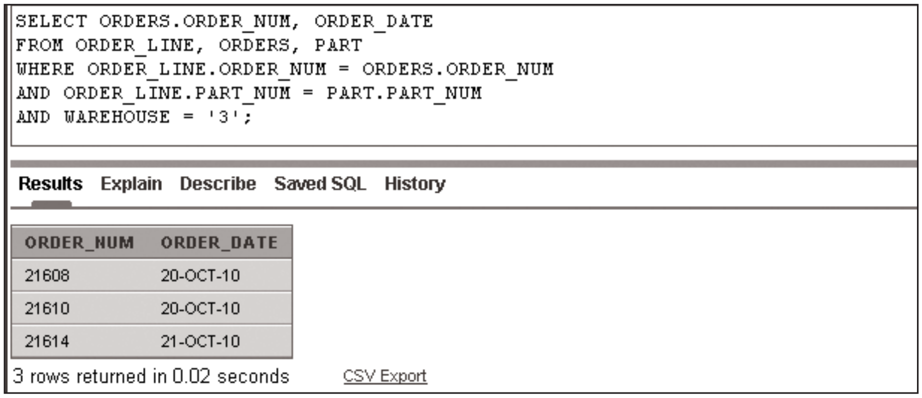
3 rows returned in 0.11 seconds

FIGURE 5-8 Nested subqueries (a subquery within a subquery)

As you might expect, SQL evaluates the queries from the innermost query to the outermost query. The query in this example is evaluated in three steps:

1. The innermost subquery is evaluated first, producing a temporary table of part numbers for those parts located in warehouse 3.
2. The next (intermediate) subquery is evaluated, producing a second temporary table with a list of order numbers. Each order number in this collection has a row in the ORDER_LINE table for which the part number is in the temporary table produced in Step 1.
3. The outer query is evaluated last, producing the desired list of order numbers and order dates. Only those orders whose numbers are in the temporary table produced in Step 2 are included in the results.

Another approach to solving Example 6 involves joining the ORDERS, ORDER_LINE, and PART tables. The query and its results appear in Figure 5-9.


```
SELECT ORDERS.ORDER_NUM, ORDER_DATE
FROM ORDER_LINE, ORDERS, PART
WHERE ORDER_LINE.ORDER_NUM = ORDERS.ORDER_NUM
AND ORDER_LINE.PART_NUM = PART.PART_NUM
AND WAREHOUSE = '3';
```

ORDER_NUM	ORDER_DATE
21608	20-OCT-10
21610	20-OCT-10
21614	21-OCT-10

3 rows returned in 0.02 seconds

FIGURE 5-9 Joining three tables

In this query, the following conditions join the tables:

```
ORDER_LINE.ORDER_NUM = ORDERS.ORDER_NUM
ORDER_LINE.PART_NUM = PART.PART_NUM
```

The condition WAREHOUSE = '3' restricts the output to only those parts located in warehouse 3.

The query results are correct regardless of which command you use. You can use whichever approach you prefer.

You might wonder whether one approach is more efficient than the other. SQL performs many built-in optimizations that analyze queries to determine the best way to satisfy them. Given a good optimizer, it should not make much difference how you formulate the query—you can see that using nested subqueries (Figure 5-8) produces the query in 0.11 seconds and joining the tables (Figure 5-9) produces the results in 0.02 seconds. If you are using a DBMS without an optimizer, however, the way you write a query *can* make a difference in the speed at which the DBMS executes the query. When you are working with a very large database and efficiency is a prime concern, consult the DBMS's manual or try some timings yourself. Try running the same query both ways to see whether you notice a difference in the speed of execution. In small databases, there should not be a significant time difference between the two approaches.

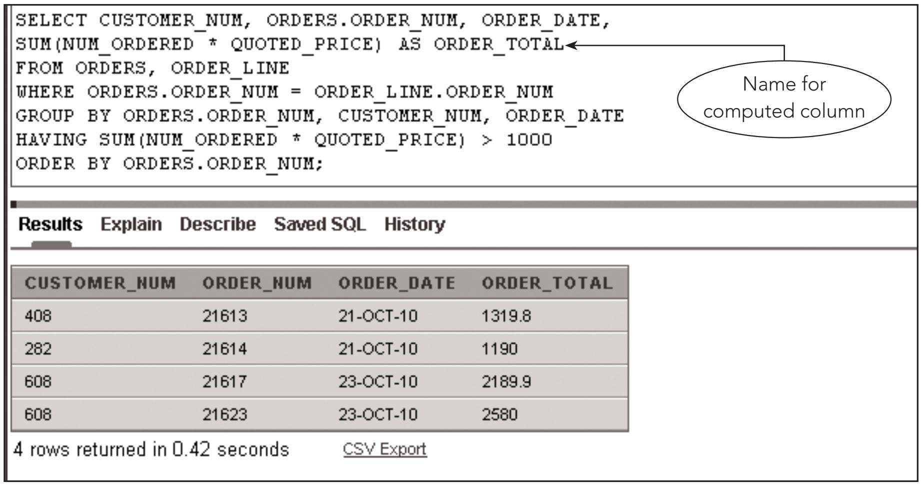
A Comprehensive Example

The query used in Example 7 involves several of the features already presented. The query illustrates all the major clauses that you can use in a SELECT command. It also illustrates the order in which these clauses must appear.

EXAMPLE 7

List the customer number, order number, order date, and order total for each order with a total that exceeds $1,000. Assign the column name ORDER_TOTAL to the column that displays order totals.

The query and its results appear in Figure 5-10.



```
SELECT CUSTOMER_NUM, ORDERS.ORDER_NUM, ORDER_DATE,
SUM(NUM_ORDERED * QUOTED_PRICE) AS ORDER_TOTAL
FROM ORDERS, ORDER_LINE
WHERE ORDERS.ORDER_NUM = ORDER_LINE.ORDER_NUM
GROUP BY ORDERS.ORDER_NUM, CUSTOMER_NUM, ORDER_DATE
HAVING SUM(NUM_ORDERED * QUOTED_PRICE) > 1000
ORDER BY ORDERS.ORDER_NUM;
```

CUSTOMER_NUM	ORDER_NUM	ORDER_DATE	ORDER_TOTAL
408	21613	21-OCT-10	1319.8
282	21614	21-OCT-10	1190
608	21617	23-OCT-10	2189.9
608	21623	23-OCT-10	2580

4 rows returned in 0.42 seconds   CSV Export

FIGURE 5-10 Comprehensive example

In this query, the ORDERS and ORDER_LINE tables are joined by listing both tables in the FROM clause and relating them in the WHERE clause. Selected data is sorted by order number using the ORDER BY clause. The GROUP BY clause indicates that the data is to be grouped by order number, customer number, and order date. For each group, the SELECT clause displays the customer number, order number, order date, and order total (SUM(NUM_ORDERED * QUOTED_PRICE)). In addition, the total was renamed ORDER_TOTAL. Not all groups will be displayed, however. The HAVING clause displays only those groups whose SUM(NUM_ORDERED * QUOTED_PRICE) is greater than $1,000.

The order number, customer number, and order date are unique for each order. Thus, it would seem that merely grouping by order number would be sufficient. SQL requires that both the customer number and the order date be listed in the GROUP BY clause. Recall that a SELECT clause can include statistics calculated for only the groups or columns whose values are identical for each row in a group. By stating that the data is to be grouped by order number, customer number, and order date, you tell SQL that the values in these columns must be the same for each row in a group.

Using an Alias

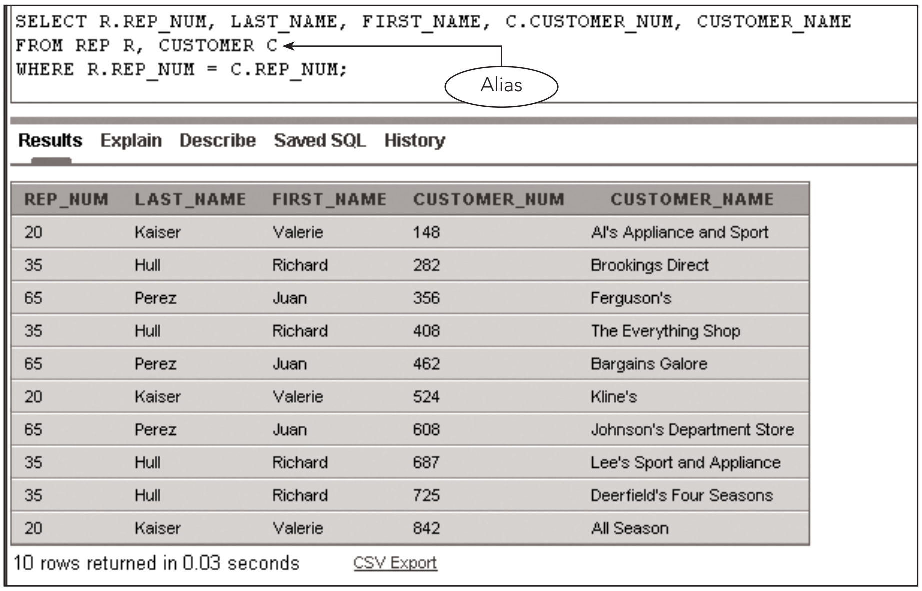
When tables are listed in the FROM clause, you can give each table an **alias**, or an alternate name, that you can use in the rest of the statement. You create an alias by typing the name of the table, pressing the Spacebar, and then typing the name of the alias. No commas or periods are necessary to separate the two names.

One reason for using an alias is simplicity. In Example 8, you assign the REP table the alias R and the CUSTOMER table the alias C. By doing this, you can type R instead of REP and C instead of CUSTOMER in the remainder of the query. The query in this example is simple, so you might not see the full benefit of this feature. When a query is complex and requires you to qualify the names, using aliases can simplify the process.

EXAMPLE 8

List the number, last name, and first name for each sales rep together with the number and name for each customer the sales rep represents.

The query and its results using aliases appear in Figure 5-11.



```
SELECT R.REP_NUM, LAST_NAME, FIRST_NAME, C.CUSTOMER_NUM, CUSTOMER_NAME
FROM REP R, CUSTOMER C
WHERE R.REP_NUM = C.REP_NUM;
```

REP_NUM	LAST_NAME	FIRST_NAME	CUSTOMER_NUM	CUSTOMER_NAME
20	Kaiser	Valerie	148	Al's Appliance and Sport
35	Hull	Richard	282	Brookings Direct
65	Perez	Juan	356	Ferguson's
35	Hull	Richard	408	The Everything Shop
65	Perez	Juan	462	Bargains Galore
20	Kaiser	Valerie	524	Kline's
65	Perez	Juan	608	Johnson's Department Store
35	Hull	Richard	687	Lee's Sport and Appliance
35	Hull	Richard	725	Deerfield's Four Seasons
20	Kaiser	Valerie	842	All Season

10 rows returned in 0.03 seconds CSV Export

FIGURE 5-11 Using aliases in a query

NOTE

Technically, it is unnecessary to qualify CUSTOMER_NUM because it is included only in the CUSTOMER table. It is qualified in Figure 5-11 for illustration purposes only.

Joining a Table to Itself

A second situation for using an alias is to join a table to itself, called a **self-join**, as illustrated in Example 9.

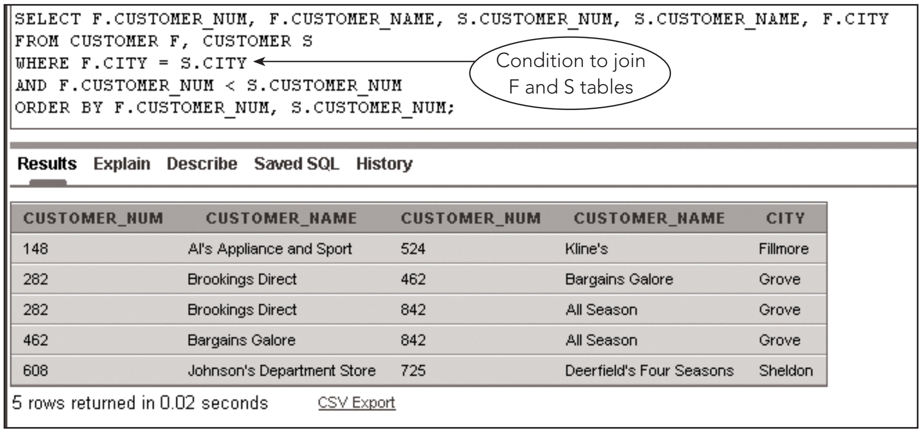
EXAMPLE 9

For each pair of customers located in the same city, display the customer number, customer name, and city.

If you had two separate tables for customers and the query requested customers in the first table having the same city as customers in the second table, you could use a normal join operation to find the answer. In this case, however, there is only *one* table (CUSTOMER) that stores all the customer information. You can treat the CUSTOMER table as if it were two tables in the query by creating an alias, as illustrated in Example 8. In this case, you use the following FROM clause:

```
FROM CUSTOMER F, CUSTOMER S
```

SQL treats this clause as a query of two tables: one that has the alias F (first), and another that has the alias S (second). The fact that both tables are really the same CUSTOMER table is not a problem. The query and its results appear in Figure 5-12.



```
SELECT F.CUSTOMER_NUM, F.CUSTOMER_NAME, S.CUSTOMER_NUM, S.CUSTOMER_NAME, F.CITY
FROM CUSTOMER F, CUSTOMER S
WHERE F.CITY = S.CITY
AND F.CUSTOMER_NUM < S.CUSTOMER_NUM
ORDER BY F.CUSTOMER_NUM, S.CUSTOMER_NUM;
```

CUSTOMER_NUM	CUSTOMER_NAME	CUSTOMER_NUM	CUSTOMER_NAME	CITY
148	Al's Appliance and Sport	524	Kline's	Fillmore
282	Brookings Direct	462	Bargains Galore	Grove
282	Brookings Direct	842	All Season	Grove
462	Bargains Galore	842	All Season	Grove
608	Johnson's Department Store	725	Deerfield's Four Seasons	Sheldon

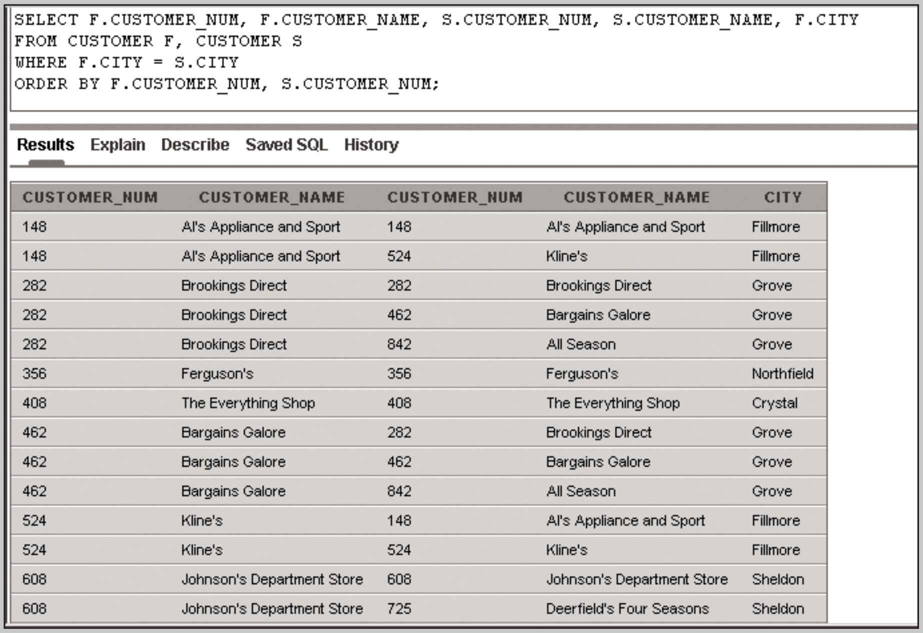
5 rows returned in 0.02 seconds

FIGURE 5-12 Using aliases for a self-join

You are requesting a customer number and name from the F table, followed by a customer number and name from the S table, and then the city. (Because the city in the first table must match the city in the second table, you can select the city from either table.) The WHERE clause contains two conditions: the cities must match, and the customer number from the first table must be less than the customer number from the second table. In addition, the ORDER BY clause ensures that the data is sorted by the first customer number. For those rows with the same first customer number, the data is further sorted by the second customer number.

Q & A

Question: Why is the condition F.CUSTOMER_NUM < S.CUSTOMER_NUM important in the query?
Answer: If you did not include this condition, you would get the query results shown in Figure 5-13.



```
SELECT F.CUSTOMER_NUM, F.CUSTOMER_NAME, S.CUSTOMER_NUM, S.CUSTOMER_NAME, F.CITY
FROM CUSTOMER F, CUSTOMER S
WHERE F.CITY = S.CITY
ORDER BY F.CUSTOMER_NUM, S.CUSTOMER_NUM;
```

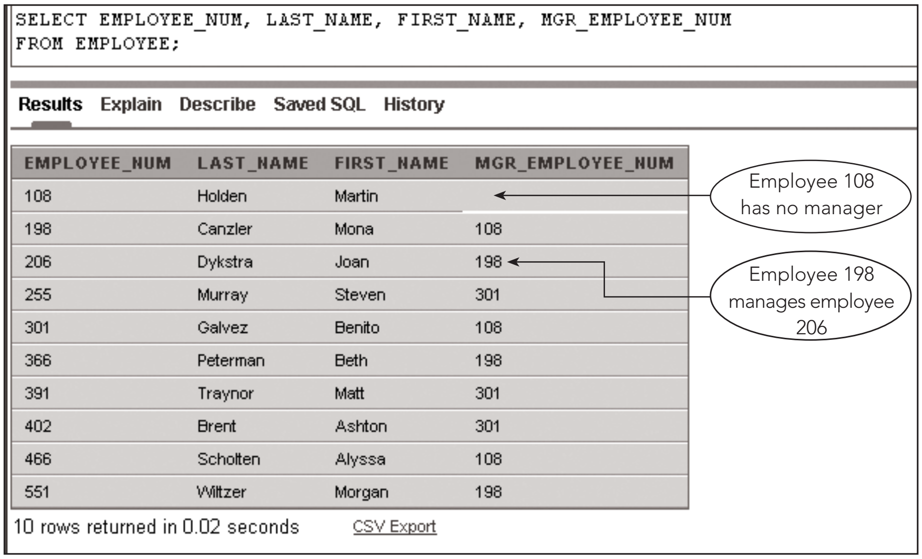
CUSTOMER_NUM	CUSTOMER_NAME	CUSTOMER_NUM	CUSTOMER_NAME	CITY
148	Al's Appliance and Sport	148	Al's Appliance and Sport	Fillmore
148	Al's Appliance and Sport	524	Kline's	Fillmore
282	Brookings Direct	282	Brookings Direct	Grove
282	Brookings Direct	462	Bargains Galore	Grove
282	Brookings Direct	842	All Season	Grove
356	Ferguson's	356	Ferguson's	Northfield
408	The Everything Shop	408	The Everything Shop	Crystal
462	Bargains Galore	282	Brookings Direct	Grove
462	Bargains Galore	462	Bargains Galore	Grove
462	Bargains Galore	842	All Season	Grove
524	Kline's	148	Al's Appliance and Sport	Fillmore
524	Kline's	524	Kline's	Fillmore
608	Johnson's Department Store	608	Johnson's Department Store	Sheldon
608	Johnson's Department Store	725	Deerfield's Four Seasons	Sheldon

FIGURE 5-13 Incorrect joining of a table to itself

The first row is included because it is true that customer number 148 (Al's Appliance and Sport) in the F table has the same city as customer number 148 (Al's Appliance and Sport) in the S table. The second row indicates that customer number 148 (Al's Appliance and Sport) has the same city as customer number 524 (Kline's). The eleventh row, however, repeats the same information because customer number 524 (Kline's) has the same city as customer number 148 (Al's Appliance and Sport). Of these three rows, the only row that should be included in the query results is the second row. The second row also is the only one of the three rows in which the first customer number (148) is less than the second customer number (524). This is why the query requires the condition F.CUSTOMER_NUM < S.CUSTOMER_NUM.

Using a Self-Join on a Primary Key Column

Figure 5-14 shows some fields from an EMPLOYEE table whose primary key is EMPLOYEE_NUM. Another field in the table is MGR_EMPLOYEE_NUM, which represents the number of the employee's manager, who also is an employee. If you look at the row for employee 206 (Joan Dykstra), you will see that employee 198 (Mona Canzler) is Joan's manager. By looking at the row for employee 198 (Mona Canzler), you see that her manager is employee 108 (Martin Holden). In the row for employee 108 (Martin Holden), the manager number is null, indicating that he has no manager.



```
SELECT EMPLOYEE_NUM, LAST_NAME, FIRST_NAME, MGR_EMPLOYEE_NUM
FROM EMPLOYEE;
```

Results
Explain
Describe
Saved SQL
History
Employee 108 has no manager
Employee 198 manages employee 206
CSV Export

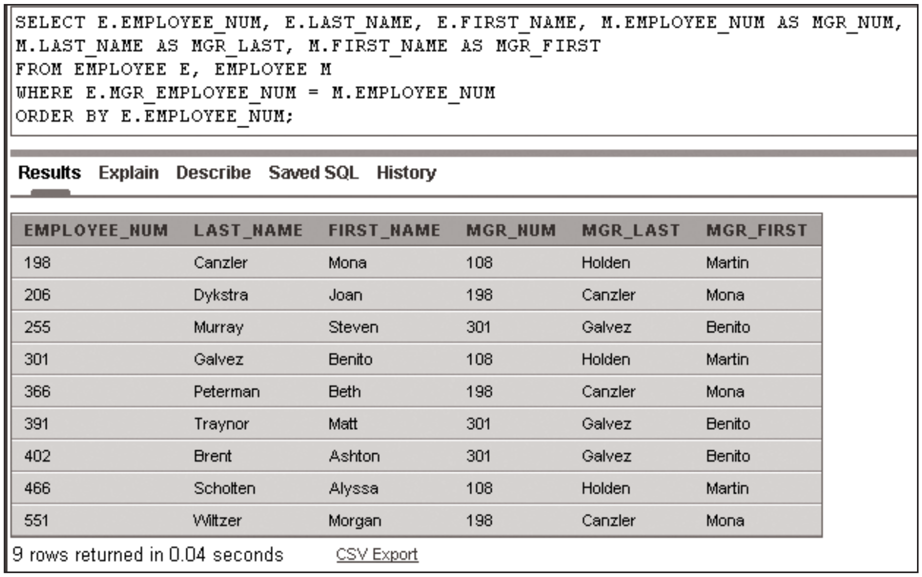
EMPLOYEE_NUM	LAST_NAME	FIRST_NAME	MGR_EMPLOYEE_NUM
108	Holden	Martin	
198	Canzler	Mona	108
206	Dykstra	Joan	198
255	Murray	Steven	301
301	Galvez	Benito	108
366	Peterman	Beth	198
391	Traynor	Matt	301
402	Brent	Ashton	301
466	Scholten	Alyssa	108
551	Wiltzer	Morgan	198

10 rows returned in 0.02 seconds

FIGURE 5-14 Employee and manager data

Suppose you need to list the employee number, employee last name, and employee first name along with the number, last name, and first name of each employee's manager. Just as in the previous self-join, you would list the EMPLOYEE table twice in the FROM clause with aliases.

The command shown in Figure 5-15 uses the letter E as an alias for the employee and the letter M as an alias for the manager. Thus E.EMPLOYEE_NUM is the employee's number and M.EMPLOYEE_NUM is the number of the employee's manager. In the SQL command, M.EMPLOYEE_NUM is renamed as MGR_NUM, M.LAST_NAME is renamed as MGR_LAST, and M.FIRST_NAME is renamed as MGR_FIRST. The condition in the WHERE clause ensures that E.MGR_EMPLOYEE_NUM (the number of the employee's manager) matches M.EMPLOYEE_NUM (the employee number on the manager's row in the table). Employee 108 is not included in the results because Martin Holden has no manager (see Figure 5-14).



```
SELECT E.EMPLOYEE_NUM, E.LAST_NAME, E.FIRST_NAME, M.EMPLOYEE_NUM AS MGR_NUM,
M.LAST_NAME AS MGR_LAST, M.FIRST_NAME AS MGR_FIRST
FROM EMPLOYEE E, EMPLOYEE M
WHERE E.MGR_EMPLOYEE_NUM = M.EMPLOYEE_NUM
ORDER BY E.EMPLOYEE_NUM;
```

EMPLOYEE_NUM	LAST_NAME	FIRST_NAME	MGR_NUM	MGR_LAST	MGR_FIRST
198	Canzler	Mona	108	Holden	Martin
206	Dykstra	Joan	198	Canzler	Mona
255	Murray	Steven	301	Galvez	Benito
301	Galvez	Benito	108	Holden	Martin
366	Peterman	Beth	198	Canzler	Mona
391	Traynor	Matt	301	Galvez	Benito
402	Brent	Ashton	301	Galvez	Benito
466	Scholten	Alyssa	108	Holden	Martin
551	Witzer	Morgan	198	Canzler	Mona

9 rows returned in 0.04 seconds

FIGURE 5-15 List of employees and their managers

Joining Several Tables

It is possible to join several tables, as illustrated in Example 10. For each pair of tables you join, you must include a condition indicating how the columns are related.

EXAMPLE 10

For each part on order, list the part number, number ordered, order number, order date, customer number, and customer name, along with the last name of the sales rep who represents each customer.

A part is on order when it occurs on any row in the ORDER_LINE table. The part number, number ordered, and order number appear in the ORDER_LINE table. If these requirements represent the entire query, you would write the query as follows:

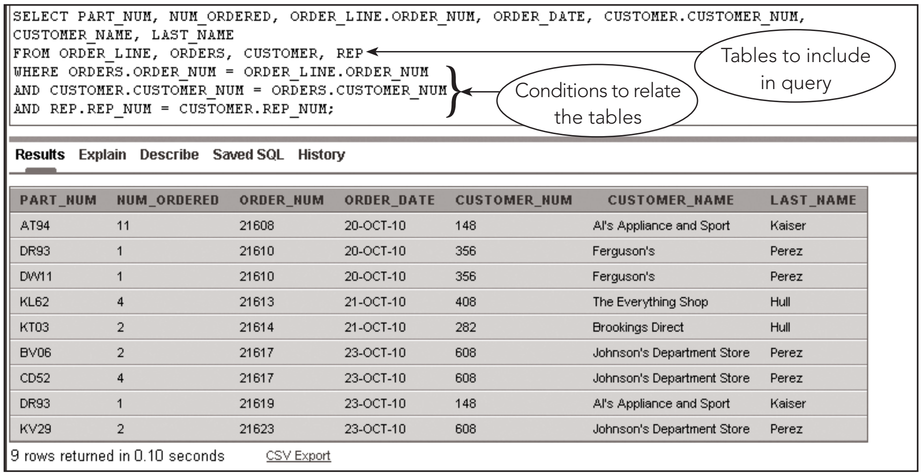
```
SELECT PART_NUM, NUM_ORDERED, ORDER_NUM
FROM ORDER_LINE;
```

This query is not sufficient, however. You also need the order date, which is in the ORDERS table; the customer number and name, which are in the CUSTOMER table; and the rep last name, which is in the REP table. Thus, you need to join *four* tables: ORDER_LINE, ORDERS, CUSTOMER, and REP. The procedure for joining more than two tables is essentially the same as the one for joining two tables. The difference is that the condition in the WHERE clause will be a compound condition. In this case, you would write the WHERE clause as follows:

```
WHERE ORDERS.ORDER_NUM = ORDER_LINE.ORDER_NUM
AND CUSTOMER.CUSTOMER_NUM = ORDERS.CUSTOMER_NUM
AND REP.REP_NUM = CUSTOMER.REP_NUM
```

The first condition relates an order to an order line with a matching order number. The second condition relates the customer to the order with a matching customer number. The final condition relates the rep to a customer with a matching sales rep number.

For the complete query, you list all the desired columns in the SELECT clause and qualify any columns that appear in more than one table. In the FROM clause, you list the tables that are involved in the query. The query and its results appear in Figure 5-16.



```
SELECT PART_NUM, NUM_ORDERED, ORDER_LINE.ORDER_NUM, ORDER_DATE, CUSTOMER.CUSTOMER_NUM,
CUSTOMER_NAME, LAST_NAME
FROM ORDER_LINE, ORDERS, CUSTOMER, REP
WHERE ORDERS.ORDER_NUM = ORDER_LINE.ORDER_NUM
AND CUSTOMER.CUSTOMER_NUM = ORDERS.CUSTOMER_NUM
AND REP.REP_NUM = CUSTOMER.REP_NUM;
```

PART_NUM	NUM_ORDERED	ORDER_NUM	ORDER_DATE	CUSTOMER_NUM	CUSTOMER_NAME	LAST_NAME
AT94	11	21608	20-OCT-10	148	Al's Appliance and Sport	Kaiser
DR93	1	21610	20-OCT-10	356	Ferguson's	Perez
DW11	1	21610	20-OCT-10	356	Ferguson's	Perez
KL62	4	21613	21-OCT-10	408	The Everything Shop	Hull
KT03	2	21614	21-OCT-10	282	Brookings Direct	Hull
BV06	2	21617	23-OCT-10	608	Johnson's Department Store	Perez
CD52	4	21617	23-OCT-10	608	Johnson's Department Store	Perez
DR93	1	21619	23-OCT-10	148	Al's Appliance and Sport	Kaiser
KV29	2	21623	23-OCT-10	608	Johnson's Department Store	Perez

9 rows returned in 0.10 seconds   CSV Export

FIGURE 5-16 Joining four tables in a query

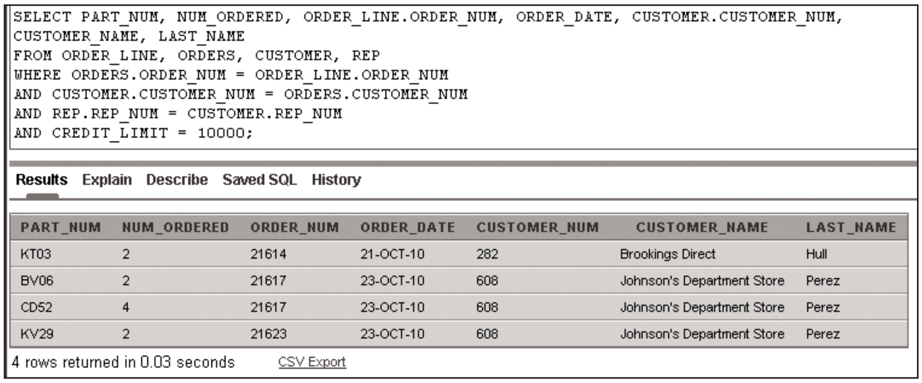
Q & A

Question: Why is the PART_NUM column, which appears in the PART and ORDER_LINE tables, not qualified in the SELECT clause?

Answer: Among the tables listed in the query, only one table contains a column named PART_NUM, so it is not necessary to qualify the table. If the PART table also appeared in the FROM clause, you would need to qualify PART_NUM to avoid confusion between the PART_NUM columns in the PART and ORDER_LINE tables.

The query shown in Figure 5-16 is more complex than many of the previous ones you have examined. You might think that SQL is not such an easy language to use after all. If you take it one step at a time, however, the query in Example 10 really is not that difficult. To construct a detailed query in a step-by-step fashion, do the following:

1. List in the SELECT clause all the columns that you want to display. If the name of a column appears in more than one table, precede the column name with the table name (that is, qualify the column name).
2. List in the FROM clause all the tables involved in the query. Usually you include the tables that contain the columns listed in the SELECT clause. Occasionally, however, there might be a table that does not contain any columns used in the SELECT clause but that does contain columns used in the WHERE clause. In this case, you also must list the table in the FROM clause. For example, if you do not need to list a customer number or name, but you do need to list the rep name, you would not include any columns from the CUSTOMER table in the SELECT clause. The CUSTOMER table still is required, however, because you must include a column from it in the WHERE clause.
3. Take one pair of related tables at a time and indicate in the WHERE clause the condition that relates the tables. Join these conditions with the AND operator. If there are any other conditions, include them in the WHERE clause and connect them to the other conditions with the AND operator. For example, if you want to view parts present on orders placed by only those customers with $10,000 credit limits, you would add one more condition to the WHERE clause, as shown in Figure 5-17.



```
SELECT PART_NUM, NUM_ORDERED, ORDER_LINE.ORDER_NUM, ORDER_DATE, CUSTOMER.CUSTOMER_NUM,
CUSTOMER_NAME, LAST_NAME
FROM ORDER_LINE, ORDERS, CUSTOMER, REP
WHERE ORDERS.ORDER_NUM = ORDER_LINE.ORDER_NUM
AND CUSTOMER.CUSTOMER_NUM = ORDERS.CUSTOMER_NUM
AND REP.REP_NUM = CUSTOMER.REP_NUM
AND CREDIT_LIMIT = 10000;
```

PART_NUM	NUM_ORDERED	ORDER_NUM	ORDER_DATE	CUSTOMER_NUM	CUSTOMER_NAME	LAST_NAME
KT03	2	21614	21-OCT-10	282	Brookings Direct	Hull
BV06	2	21617	23-OCT-10	608	Johnson's Department Store	Perez
CD52	4	21617	23-OCT-10	608	Johnson's Department Store	Perez
KV29	2	21623	23-OCT-10	608	Johnson's Department Store	Perez

4 rows returned in 0.03 seconds   CSV Export

FIGURE 5-17 Restricting the rows when joining four tables

SET OPERATIONS

In SQL, you can use the set operations for taking the union, intersection, and difference of two tables. The **union** of two tables uses the **UNION** operator to create a temporary table containing every row that is in either the first table, the second table, or both tables. The **intersection** of two tables uses the **INTERSECT** operator to create a temporary table containing all rows that are in both tables. The **difference** of two tables uses the **MINUS** operator to create a temporary table containing the set of all rows that are in the first table but that are not in the second table.

For example, suppose that TEMP1 is a table containing the number and name of each customer represented by sales rep 65. Further suppose that TEMP2 is a table containing the number and name of those customers that currently have orders on file, as shown in Figure 5-18.

TEMP1

CUSTOMER_NUM	CUSTOMER_NAME
356	Ferguson's
462	Bargains Galore
608	Johnson's Department Store

TEMP2

CUSTOMER_NUM	CUSTOMER_NAME
148	Al's Appliance and Sport
282	Brookings Direct
356	Ferguson's
408	The Everything Shop
608	Johnson's Department Store

FIGURE 5-18 Customers of rep 65 and customers with open orders

The union of TEMP1 and TEMP2 (TEMP1 UNION TEMP2) consists of the number and name of those customers that are represented by sales rep 65 *or* that currently have orders on file, *or* both. The intersection of these two tables (TEMP1 INTERSECT TEMP2) contains those customers that are represented by sales rep 65 *and* that have orders on file. The difference of these two tables (TEMP1 MINUS TEMP2) contains those customers that are represented by sales rep 65 but that *do not* have orders on file. The results of these set operations are shown in Figure 5-19.

TEMP1 UNION TEMP2

CUSTOMER_NUM	CUSTOMER_NAME
148	Al's Appliance and Sport
282	Brookings Direct
356	Ferguson's
408	The Everything Shop
462	Bargains Galore
608	Johnson's Department Store

TEMP1 INTERSECT TEMP2

CUSTOMER_NUM	CUSTOMER_NAME
356	Ferguson's
608	Johnson's Department Store

TEMP1 MINUS TEMP2

CUSTOMER_NUM	CUSTOMER_NAME
462	Bargains Galore

FIGURE 5-19 Union, intersection, and difference of the TEMP1 and TEMP2 tables

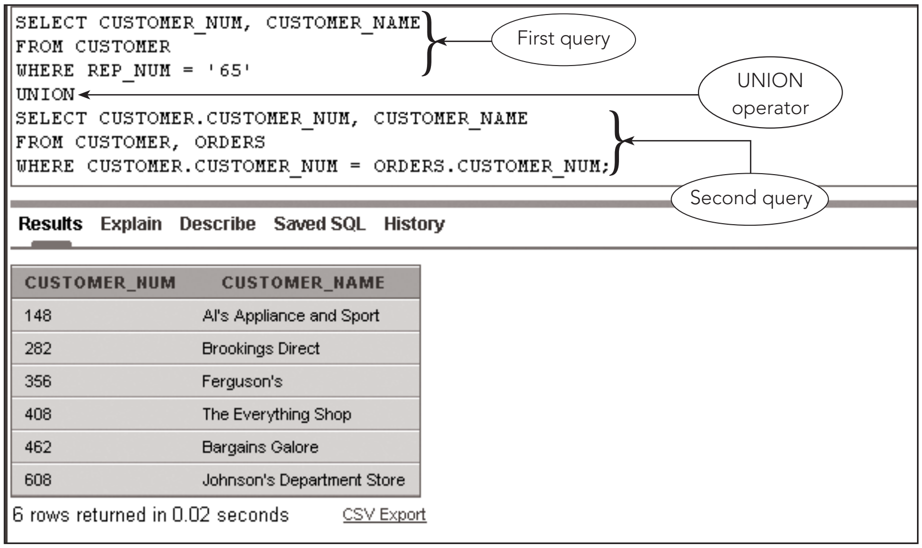
There is a restriction on set operations. It does not make sense, for example, to talk about the union of the CUSTOMER table and the ORDERS table because these tables do not contain the same columns. What might rows in this union look like? The two tables in the union *must* have the same structure for a union to be appropriate; the formal term is "union compatible." Two tables are **union compatible** when they have the same number of columns and their corresponding columns have identical data types and lengths.

Note that the definition of union compatible does not state that the columns of the two tables must be identical but rather that the columns must be of the same type. Thus, if one column is CHAR(20), the matching column also must be CHAR(20).

EXAMPLE 11

List the number and name of each customer that either is represented by sales rep 65 or that currently has orders on file, or both.

You can create a temporary table containing the number and name of each customer that is represented by sales rep 65 by selecting the customer numbers and names from the CUSTOMER table for which the sales rep number is 65. Then you can create another temporary table containing the number and name of each customer that currently has orders on file by joining the CUSTOMER and ORDERS tables. The two temporary tables created by this process have the same structure; that is, they both contain the CUSTOMER_NUM and CUSTOMER_NAME columns. Because the temporary tables are union compatible, it is possible to take the union of these two tables. The query and its results appear in Figure 5-20.



```
SELECT CUSTOMER_NUM, CUSTOMER_NAME
FROM CUSTOMER
WHERE REP_NUM = '65'
UNION
SELECT CUSTOMER.CUSTOMER_NUM, CUSTOMER_NAME
FROM CUSTOMER, ORDERS
WHERE CUSTOMER.CUSTOMER_NUM = ORDERS.CUSTOMER_NUM;
```

CUSTOMER_NUM	CUSTOMER_NAME
148	Al's Appliance and Sport
282	Brookings Direct
356	Ferguson's
408	The Everything Shop
462	Bargains Galore
608	Johnson's Department Store

6 rows returned in 0.02 seconds

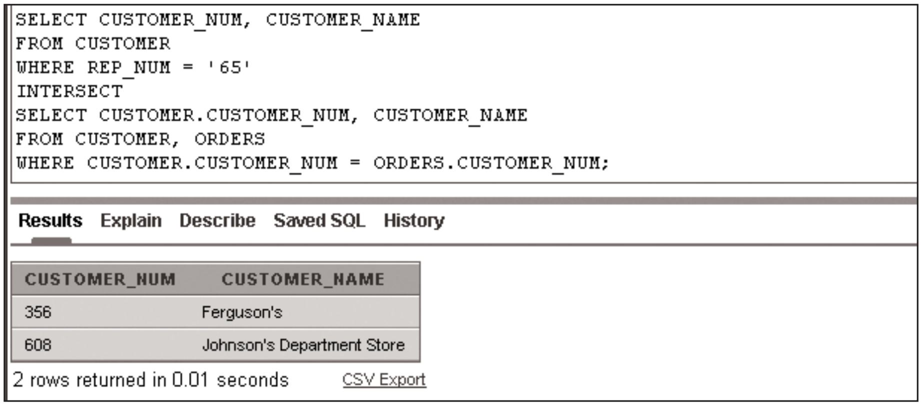
FIGURE 5-20 Using the UNION operator

If your SQL implementation truly supports the union operation, it will remove any duplicate rows automatically. For example, any customer that is represented by sales rep 65 *and* that currently has orders on file will appear only once in the results. Oracle, Access, and SQL Server support the union operation and correctly remove duplicates.

EXAMPLE 12

List the number and name of each customer that is represented by sales rep 65 and that currently has orders on file.

The only difference between this query and the one in Example 11 is that the appropriate operator to use is INTERSECT, as shown in Figure 5-21.



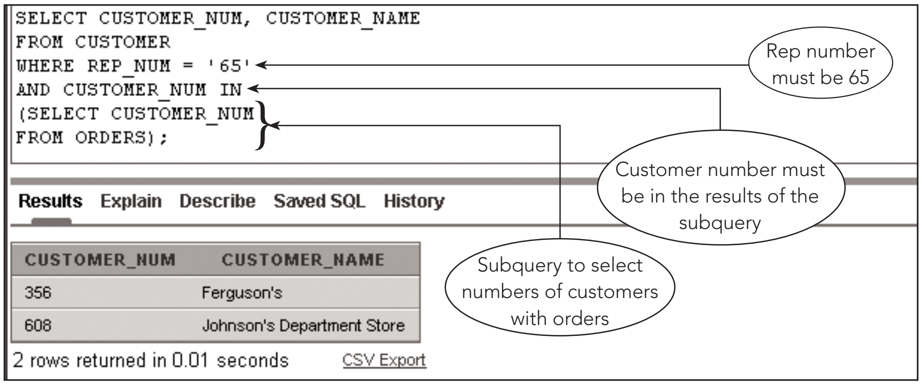
```
SELECT CUSTOMER_NUM, CUSTOMER_NAME
FROM CUSTOMER
WHERE REP_NUM = '65'
INTERSECT
SELECT CUSTOMER.CUSTOMER_NUM, CUSTOMER_NAME
FROM CUSTOMER, ORDERS
WHERE CUSTOMER.CUSTOMER_NUM = ORDERS.CUSTOMER_NUM;
```

CUSTOMER_NUM	CUSTOMER_NAME
356	Ferguson's
608	Johnson's Department Store

2 rows returned in 0.01 seconds

FIGURE 5-21 Using the INTERSECT operator

Some SQL implementations do not support the INTERSECT operator, so you need to take a different approach. The command shown in Figure 5-22 produces the same results as the INTERSECT operator by using the IN operator and a subquery. The command selects the number and name of each customer that is represented by sales rep 65 and whose customer number also appears in the collection of customer numbers in the ORDERS table.



```
SELECT CUSTOMER_NUM, CUSTOMER_NAME
FROM CUSTOMER
WHERE REP_NUM = '65'
AND CUSTOMER_NUM IN
(SELECT CUSTOMER_NUM
FROM ORDERS);
```

Rep number must be 65
Customer number must be in the results of the subquery
Subquery to select numbers of customers with orders
Results Explain Describe Saved SQL History

CUSTOMER_NUM	CUSTOMER_NAME
356	Ferguson's
608	Johnson's Department Store

2 rows returned in 0.01 seconds

FIGURE 5-22 Performing an intersection without using the INTERSECT operator

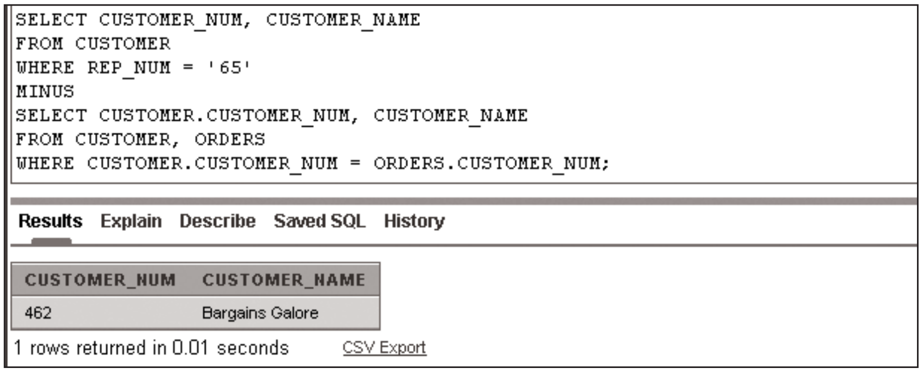
NOTE

Oracle and SQL Server support the INTERSECT operator but Microsoft Access does not.

EXAMPLE 13

List the number and name of each customer that is represented by sales rep 65 but that does not have orders currently on file.

The query uses the MINUS operator, as shown in Figure 5-23.



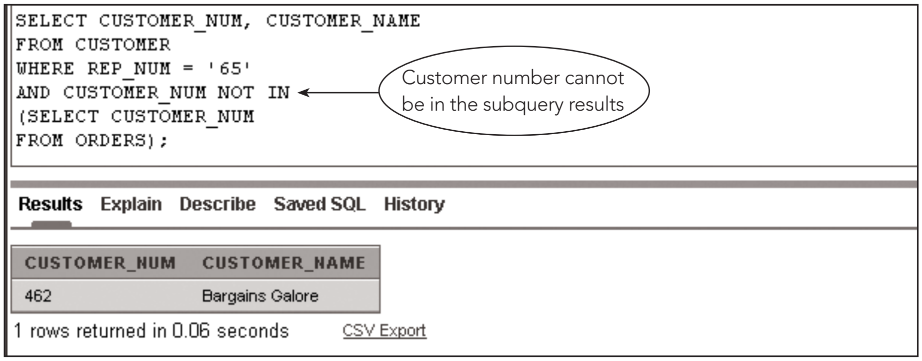
```
SELECT CUSTOMER_NUM, CUSTOMER_NAME
FROM CUSTOMER
WHERE REP_NUM = '65'
MINUS
SELECT CUSTOMER.CUSTOMER_NUM, CUSTOMER_NAME
FROM CUSTOMER, ORDERS
WHERE CUSTOMER.CUSTOMER_NUM = ORDERS.CUSTOMER_NUM;
```

CUSTOMER_NUM	CUSTOMER_NAME
462	Bargains Galore

```
1 rows returned in 0.01 seconds
```

FIGURE 5-23 Using the MINUS operator

Just as with the INTERSECT operator, some SQL implementations do not support the MINUS operator. In such cases, you need to take a different approach, such as the one shown in Figure 5-24. This command produces the same results by selecting the number and name of each customer that is represented by sales rep 65 and whose customer number does *not* appear in the collection of customer numbers in the ORDERS table.



```
SELECT CUSTOMER_NUM, CUSTOMER_NAME
FROM CUSTOMER
WHERE REP_NUM = '65'
AND CUSTOMER_NUM NOT IN
(SELECT CUSTOMER_NUM
FROM ORDERS);
```

CUSTOMER_NUM	CUSTOMER_NAME
462	Bargains Galore

```
1 rows returned in 0.06 seconds
```

FIGURE 5-24 Performing a difference without using the MINUS operator

NOTE

Oracle supports the MINUS operator, but SQL Server and Microsoft Access do not.

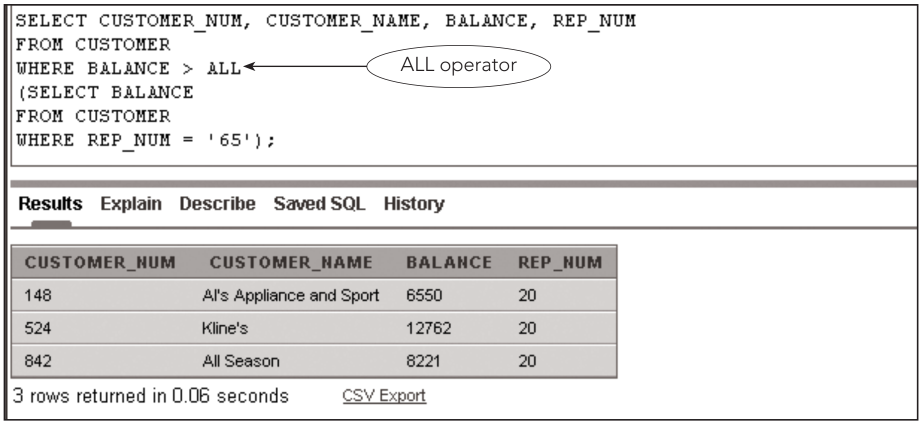
ALL AND ANY

You can use the ALL and ANY operators with subqueries to produce a single column of numbers. When you precede the subquery by the **ALL** operator, the condition is true only if it satisfies *all* values produced by the subquery. When you precede the subquery by the **ANY** operator, the condition is true only if it satisfies *any* value (one or more) produced by the subquery. The following examples illustrate the use of these operators.

EXAMPLE 14

Find the customer number, name, current balance, and rep number of each customer whose balance exceeds the maximum balance of all customers represented by sales rep 65.

You can find the maximum balance of the customers represented by sales rep 65 in a subquery and then find all customers whose balances are greater than this number. There is an alternative method that is simpler, however. You can use the ALL operator, as shown in Figure 5-25.



```
SELECT CUSTOMER_NUM, CUSTOMER_NAME, BALANCE, REP_NUM
FROM CUSTOMER
WHERE BALANCE > ALL
(SELECT BALANCE
FROM CUSTOMER
WHERE REP_NUM = '65');
```

CUSTOMER_NUM	CUSTOMER_NAME	BALANCE	REP_NUM
148	Al's Appliance and Sport	6550	20
524	Kline's	12762	20
842	All Season	8221	20

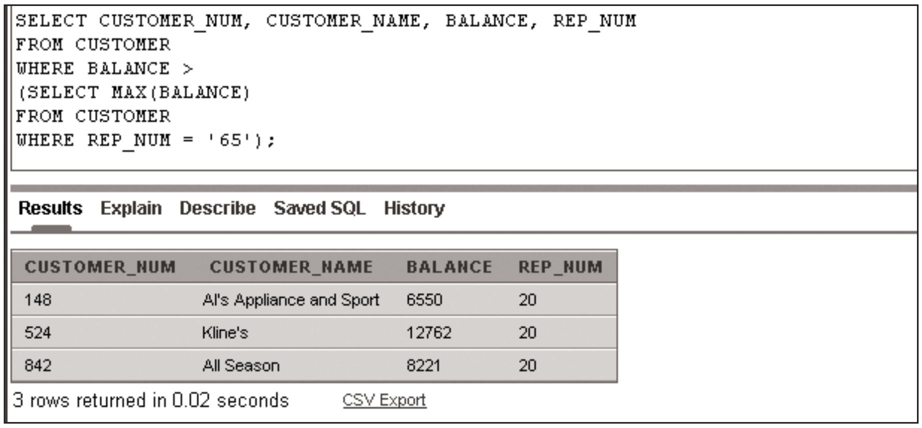
3 rows returned in 0.06 seconds CSV Export

FIGURE 5-25 SELECT command that uses the ALL operator

To some users, the query shown in Figure 5-25 might seem more natural than finding the maximum balance in the subquery. For other users, the opposite might be true. You can use whichever approach you prefer.

Q & A

Question: How would you get the same result for Example 14 without using the ALL operator?
Answer: You could select each customer whose balance is greater than the maximum balance of any customer of sales rep 65, as shown in Figure 5-26.



```
SELECT CUSTOMER_NUM, CUSTOMER_NAME, BALANCE, REP_NUM
FROM CUSTOMER
WHERE BALANCE >
(SELECT MAX(BALANCE)
FROM CUSTOMER
WHERE REP_NUM = '65');
```

CUSTOMER_NUM	CUSTOMER_NAME	BALANCE	REP_NUM
148	Al's Appliance and Sport	6550	20
524	Kline's	12762	20
842	All Season	8221	20

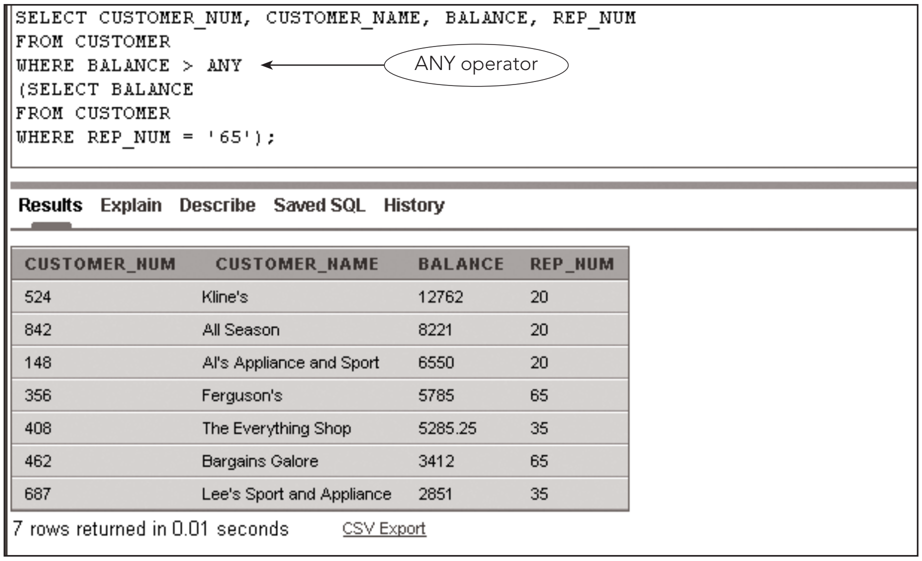
3 rows returned in 0.02 seconds

FIGURE 5-26 Alternative to using the ALL operator

EXAMPLE 15

Find the customer number, name, current balance, and rep number of each customer whose balance is greater than the balance of at least one customer of sales rep 65.

You can find the minimum balance of the customers represented by sales rep 65 in a subquery and then find all customers whose balance is greater than this number. To simplify the process, you can use the ANY operator, as shown in Figure 5-27.



```
SELECT CUSTOMER_NUM, CUSTOMER_NAME, BALANCE, REP_NUM
FROM CUSTOMER
WHERE BALANCE > ANY
(SELECT BALANCE
FROM CUSTOMER
WHERE REP_NUM = '65');
```

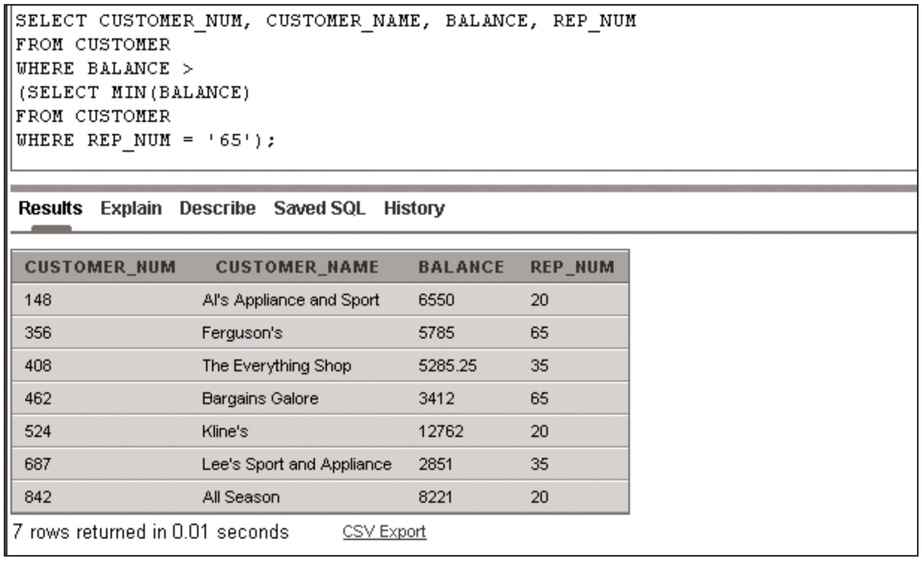
CUSTOMER_NUM	CUSTOMER_NAME	BALANCE	REP_NUM
524	Kline's	12762	20
842	All Season	8221	20
148	Al's Appliance and Sport	6550	20
356	Ferguson's	5785	65
408	The Everything Shop	5285.25	35
462	Bargains Galore	3412	65
687	Lee's Sport and Appliance	2851	35

```
7 rows returned in 0.01 seconds
```

FIGURE 5-27 SELECT command with an ANY operator

Q & A

Question: How would you get the same results without using the ANY operator?
Answer: You could select each customer whose balance is greater than the minimum balance of any customer of sales rep 65, as shown in Figure 5-28.



```
SELECT CUSTOMER_NUM, CUSTOMER_NAME, BALANCE, REP_NUM
FROM CUSTOMER
WHERE BALANCE >
(SELECT MIN(BALANCE)
FROM CUSTOMER
WHERE REP_NUM = '65');
```

CUSTOMER_NUM	CUSTOMER_NAME	BALANCE	REP_NUM
148	Al's Appliance and Sport	6550	20
356	Ferguson's	5785	65
408	The Everything Shop	5285.25	35
462	Bargains Galore	3412	65
524	Kline's	12762	20
687	Lee's Sport and Appliance	2851	35
842	All Season	8221	20

7 rows returned in 0.01 seconds

FIGURE 5-28 Alternative to using the ANY operator

SPECIAL OPERATIONS

You can perform special operations within SQL, such as the self-join that you already used. Three other special operations are the inner join, the outer join, and the product.

Inner Join

A join that compares the tables in a FROM clause and lists only those rows that satisfy the condition in the WHERE clause is called an **inner join**. The joins that you have performed so far in this text have been inner joins. Example 16 illustrates the inner join.

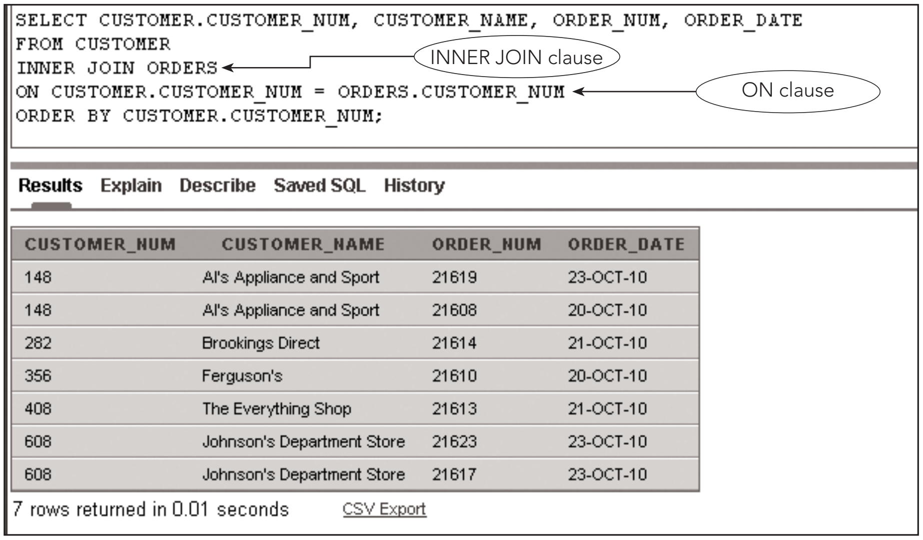
EXAMPLE 16

Display the customer number, customer name, order number, and order date for each order. Sort the results by customer number.

This example requires the same type of join that you have been using. The command is:

```
SELECT CUSTOMER.CUSTOMER_NUM, CUSTOMER_NAME,
     ORDER_NUM, ORDER_DATE
FROM CUSTOMER, ORDERS
WHERE CUSTOMER.CUSTOMER_NUM = ORDERS.CUSTOMER_NUM
ORDER BY CUSTOMER.CUSTOMER_NUM;
```

The previous approach should work in any SQL implementation. An update to the SQL standard approved in 1992, called SQL-92, provides an alternative way of performing an inner join, as demonstrated in Figure 5-29.



```
SELECT CUSTOMER.CUSTOMER_NUM, CUSTOMER_NAME, ORDER_NUM, ORDER_DATE
FROM CUSTOMER
INNER JOIN ORDERS
ON CUSTOMER.CUSTOMER_NUM = ORDERS.CUSTOMER_NUM
ORDER BY CUSTOMER.CUSTOMER_NUM;
```

CUSTOMER_NUM	CUSTOMER_NAME	ORDER_NUM	ORDER_DATE
148	Al's Appliance and Sport	21619	23-OCT-10
148	Al's Appliance and Sport	21608	20-OCT-10
282	Brookings Direct	21614	21-OCT-10
356	Ferguson's	21610	20-OCT-10
408	The Everything Shop	21613	21-OCT-10
608	Johnson's Department Store	21623	23-OCT-10
608	Johnson's Department Store	21617	23-OCT-10

```
7 rows returned in 0.01 seconds
```

FIGURE 5-29 Query that uses an INNER JOIN clause

In the FROM clause, list the first table, and then include an INNER JOIN clause that includes the name of the second table. Instead of a WHERE clause, use an ON clause containing the same condition that you would have included in the WHERE clause.

Outer Join

Sometimes you need to list all the rows from one of the tables in a join, regardless of whether they match any rows in a second table. For example, you can perform the join of the CUSTOMER and ORDERS tables in the query for Example 16, but display all customers—even the ones without orders. This type of join is called an **outer join**.

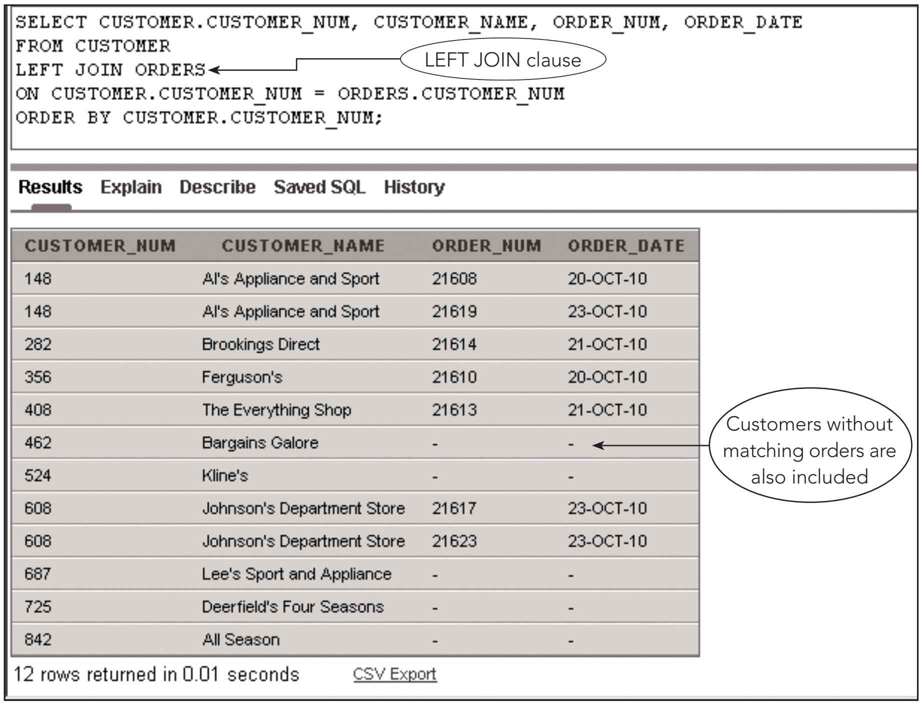
There are actually three types of outer joins. In a **left outer join**, all rows from the table on the left (the table listed first in the query) are included regardless of whether they match rows from the table on the right (the table listed second in the query). Rows from the table on the right are included only when they match. In a **right outer join**, all rows from the table on the right are included regardless of whether they match rows from the table on the left. Rows from the table on the left are included only when they match. In a **full outer join**, all rows from both tables are included regardless of whether they match rows from the other table. (The full outer join is rarely used.)

Example 17 illustrates the use of a left outer join.

EXAMPLE 17

Display the customer number, customer name, order number, and order date for all orders. Include all customers in the results. For customers that do not have orders, omit the order number and order date.

To include all customers, you must perform an outer join. Assuming the CUSTOMER table is listed first, the join should be a left outer join. In SQL, you use the LEFT JOIN clause to perform a left outer join as shown in Figure 5-30. (You would use a RIGHT JOIN clause to perform a right outer join.)



```
SELECT CUSTOMER.CUSTOMER_NUM, CUSTOMER_NAME, ORDER_NUM, ORDER_DATE
FROM CUSTOMER
LEFT JOIN ORDERS
ON CUSTOMER.CUSTOMER_NUM = ORDERS.CUSTOMER_NUM
ORDER BY CUSTOMER.CUSTOMER_NUM;
```

LEFT JOIN clause
Results Explain Describe Saved SQL History
Customers without matching orders are also included

CUSTOMER_NUM	CUSTOMER_NAME	ORDER_NUM	ORDER_DATE
148	Al's Appliance and Sport	21608	20-OCT-10
148	Al's Appliance and Sport	21619	23-OCT-10
282	Brookings Direct	21614	21-OCT-10
356	Ferguson's	21610	20-OCT-10
408	The Everything Shop	21613	21-OCT-10
462	Bargains Galore	-	-
524	Kline's	-	-
608	Johnson's Department Store	21617	23-OCT-10
608	Johnson's Department Store	21623	23-OCT-10
687	Lee's Sport and Appliance	-	-
725	Deerfield's Four Seasons	-	-
842	All Season	-	-

```
12 rows returned in 0.01 seconds      CSV Export
```

FIGURE 5-30 Query that uses a LEFT JOIN clause

All customers are included in the results. For customers without orders, the order number and date are blank. Technically, these blank values are null.

NOTE

In Oracle, there is another way to perform left and right outer joins. You write the join as you have been doing, with one exception. You include parentheses and a plus sign in the WHERE clause after the column in the table for which only matching rows are to be included. In this example, the plus sign would follow the CUSTOMER_NUM column in the ORDERS table because only orders that match customers are to be included. Because customers that do not have orders are to be included in the results, there would be no plus sign after the CUSTOMER_NUM column in the CUSTOMER table. The correct query is as follows:

```
SELECT CUSTOMER.CUSTOMER_NUM, CUSTOMER_NAME,
ORDER_NUM, ORDER_DATE
FROM CUSTOMER, ORDERS
WHERE CUSTOMER.CUSTOMER_NUM = ORDERS.CUSTOMER_NUM(+)
ORDER BY CUSTOMER.CUSTOMER_NUM;
```

Running this query produces the same results shown in Figure 5-30.

Product

The **product** (formally called the **Cartesian product**) of two tables is the combination of all rows in the first table and all rows in the second table.

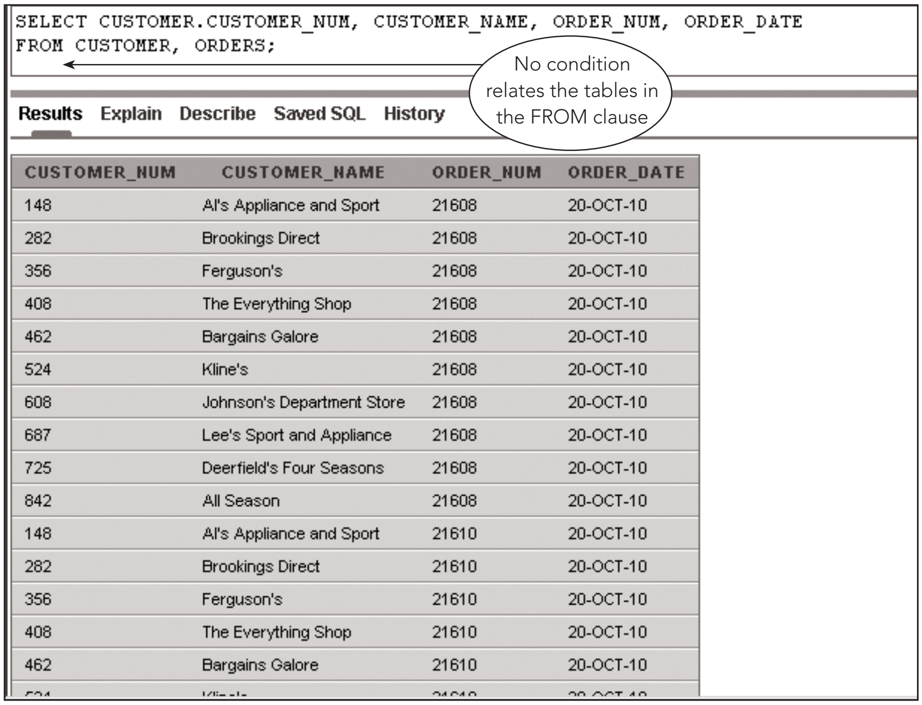
NOTE

The product operation is not common. You need to be aware of it, however, because it is easy to create a product inadvertently by omitting the WHERE clause when you are attempting to join tables.

EXAMPLE 18

Form the product of the CUSTOMER and ORDERS tables. Display the customer number and name from the CUSTOMER table, along with the order number and order date from the ORDERS table.

Forming a product is actually very easy. You simply omit the WHERE clause, as shown in Figure 5-31.



```
SELECT CUSTOMER.CUSTOMER_NUM, CUSTOMER_NAME, ORDER_NUM, ORDER_DATE
FROM CUSTOMER, ORDERS;
```

CUSTOMER_NUM	CUSTOMER_NAME	ORDER_NUM	ORDER_DATE
148	Al's Appliance and Sport	21608	20-OCT-10
282	Brookings Direct	21608	20-OCT-10
356	Ferguson's	21608	20-OCT-10
408	The Everything Shop	21608	20-OCT-10
462	Bargains Galore	21608	20-OCT-10
524	Kline's	21608	20-OCT-10
608	Johnson's Department Store	21608	20-OCT-10
687	Lee's Sport and Appliance	21608	20-OCT-10
725	Deerfield's Four Seasons	21608	20-OCT-10
842	All Season	21608	20-OCT-10
148	Al's Appliance and Sport	21610	20-OCT-10
282	Brookings Direct	21610	20-OCT-10
356	Ferguson's	21610	20-OCT-10
408	The Everything Shop	21610	20-OCT-10
462	Bargains Galore	21610	20-OCT-10

FIGURE 5-31 Query that produces a product of two tables

Q & A

Question: Figure 5-31 does not show all the rows in the result. How many rows are actually included?
Answer: The CUSTOMER table has 10 rows and the ORDERS table has seven rows. Because each of the 10 customer rows is matched with each of the seven order rows, there are 70 (10 x 7) rows in the result.

Chapter Summary

- To join tables, indicate in the SELECT clause all columns to display, list in the FROM clause all tables to join, and then include in the WHERE clause any conditions requiring values in matching columns to be equal.
- When referring to matching columns in different tables, you must qualify the column names to avoid confusion. You qualify column names using the following format: table name.column name.
- Use the IN or EXISTS operators with an appropriate subquery as an alternate way of performing a join.
- A subquery can contain another subquery. The innermost subquery is executed first.
- The name of a table in a FROM clause can be followed by an alias, which is an alternate name for a table. You can use the alias in place of the table name throughout the SQL command. By using two different aliases for the same table in a single SQL command, you can join a table to itself.
- The UNION operator creates a union of two tables (the collection of rows that are in either or both tables). The INTERSECT operator creates the intersection of two tables (the collection of rows that are in both tables). The MINUS operator creates the difference of two tables (the collection of rows that are in the first table but not in the second table). To perform any of these operations, the tables involved must be union compatible. Two tables are union compatible when they have the same number of columns and their corresponding columns have identical data types and lengths.
- When the ALL operator precedes a subquery, the condition is true only if it is satisfied by *all* values produced by the subquery.
- When the ANY operator precedes a subquery, the condition is true only if it is satisfied by *any* value (one or more) produced by the subquery.
- In an inner join, only matching rows from both tables are included. You can use the INNER JOIN clause to perform an inner join.
- In a left outer join, all rows from the table on the left (the table listed first in the query) are included regardless of whether they match rows from the table on the right (the table listed second in the query). Rows from the table on the right are included only when they match. You can use the LEFT JOIN clause to perform a left outer join. In a right outer join, all rows from the table on the right are included regardless of whether they match rows from the table on the left. Rows from the table on the left are included only when they match. You can use the RIGHT JOIN clause to perform a right outer join.
- The product (Cartesian product) of two tables is the combination of all rows in the first table and all rows in the second table. To form a product of two tables, include both tables in the FROM clause and omit the WHERE clause.

Key Terms

alias	join
ALL	left outer join
ANY	MINUS
Cartesian product	nested subquery
correlated subquery	outer join
difference	product
EXISTS	right outer join
full outer join	self-join
inner join	union
INTERSECT	UNION
intersection	union compatible

Review Questions

1. How do you join tables in SQL?
2. When must you qualify names in SQL commands? How do you qualify a column name?
3. List two operators that you can use with subqueries as an alternate way of performing joins.
4. What is a nested subquery? In which order does SQL evaluate nested subqueries?
5. What is an alias? How do you specify an alias in SQL? How do you use an alias?
6. How do you join a table to itself in SQL?
7. How do you take the union of two tables in SQL? How do you take the intersection of two tables in SQL? How do you take the difference of two tables in SQL? Are there any restrictions on the tables when performing any of these operations?
8. What does it mean for two tables to be union compatible?
9. How do you use the ALL operator with a subquery?
10. How do you use the ANY operator with a subquery?
11. Which rows are included in an inner join? What clause can you use to perform an inner join in SQL?
12. Which rows are included in a left outer join? What clause can you use to perform a left outer join in SQL?
13. Which rows are included in a right outer join? What clause can you use to perform a right outer join in SQL?
14. What is the formal name for the product of two tables? How do you form a product in SQL?
15. Use your favorite Web browser and Web search engine to find definitions for the terms equi-join, natural join, and cross join. Write a short report that identifies how these terms relate to the terms join, inner join, and Cartesian product. Be sure to reference your online sources properly.

16. Use your favorite Web browser and Web search engine to find information on cost-based query optimizers. Write a short report that explains how cost-based query optimization works, and what type(s) of queries benefit the most from cost-based query optimization. Be sure to reference your online sources properly.

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. For each order, list the order number and order date along with the number and name of the customer that placed the order.
2. For each order placed on October 23, 2010, list the order number along with the number and name of the customer that placed the order.
3. For each order, list the order number, order date, part number, number of units ordered, and quoted price for each order line that makes up the order.
4. Use the IN operator to find the number and name of each customer that placed an order on October 23, 2010.
5. Repeat Exercise 4, but this time use the EXISTS operator in your answer.
6. Find the number and name of each customer that did not place an order on October 23, 2010.
7. For each order, list the order number, order date, part number, part description, and item class for each part that makes up the order.
8. Repeat Exercise 7, but this time order the rows by item class and then by order number.
9. Use a subquery to find the rep number, last name, and first name of each sales rep who represents at least one customer with a credit limit of $10,000. List each sales rep only once in the results.
10. Repeat Exercise 9, but this time do not use a subquery.
11. Find the number and name of each customer that currently has an order on file for a Gas Range.
12. List the part number, part description, and item class for each pair of parts that are in the same item class. (For example, one such pair would be part AT94 and part FD21, because the item class for both parts is HW.)
13. List the order number and order date for each order placed by the customer named Johnson's Department Store. (*Hint:* To enter an apostrophe (single quotation mark) within a string of characters, type two single quotation marks.)
14. List the order number and order date for each order that contains an order line for an Iron.
15. List the order number and order date for each order that either was placed by Johnson's Department Store or that contains an order line for a Gas Range.
16. List the order number and order date for each order that was placed by Johnson's Department Store and that contains an order line for a Gas Range.

17. List the order number and order date for each order that was placed by Johnson's Department Store but that does not contain an order line for a Gas Range.
18. List the part number, part description, unit price, and item class for each part that has a unit price greater than the unit price of every part in item class AP. Use either the ALL or ANY operator in your query. (*Hint:* Make sure you select the correct operator.)
19. If you used ALL in Exercise 18, repeat the exercise using ANY. If you used ANY, repeat the exercise using ALL, and then run the new command. What question does this command answer?
20. For each part, list the part number, description, units on hand, order number, and number of units ordered. All parts should be included in the results. For those parts that are currently not on order, the order number and number of units ordered should be left blank. Order the results by part number.

Henry Books

Use SQL and the Henry Books database (see 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. For each book, list the book code, book title, publisher code, and publisher name. Order the results by publisher name.
2. For each book published by Scribner, list the book code, book title, and price.
3. List the book title, book code, and price of each book published by Scribner that has a book price of at least $14.
4. List the book code, book title, and units on hand for each book in branch number 3.
5. List the book title for each book that has the type PSY and that is published by Berkley Publishing.
6. Find the book title for each book written by author number 18. Use the IN operator in your query.
7. Repeat Exercise 6, but this time use the EXISTS operator in your query.
8. Find the book code and book title for each book located in branch number 2 and written by author 20.
9. List the book codes for each pair of books that have the same price. (For example, one such pair would be book 0200 and book 7559, because the price of both books is $8.00.) The first book code listed should be the major sort key, and the second book code should be the minor sort key.
10. Find the book title, author last name, and units on hand for each book in branch number 4.
11. Repeat Exercise 10, but this time list only paperback books.
12. Find the book code and book title for each book whose price is more than $10 or that was published in Boston.
13. Find the book code and book title for each book whose price is more than $10 and that was published in Boston.
14. Find the book code and book title for each book whose price is more than $10 but that was not published in Boston.

15. Find the book code and book title for each book whose price is greater than the book price of every book that has the type MYS.
16. Find the book code and book title for each book whose price is greater than the price of at least one book that has the type MYS.
17. List the book code, book title, and units on hand for each book in branch number 2. Be sure each book is included, regardless of whether there are any copies of the book currently on hand in branch 2. Order the output by book code.

Alexamara Marina Group

Use SQL and the Alexamara Marina Group database (see 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. For every boat, list the marina number, slip number, boat name, owner number, owner's first name, and owner's last name.
2. For every completed or open service request for routine engine maintenance, list the slip ID, description, and status.
3. For every service request for routine engine maintenance, list the slip ID, marina number, slip number, estimated hours, spent hours, owner number, and owner's last name.
4. List the first and last names of all owners who have a boat in a 30-foot slip. Use the IN operator in your query.
5. Repeat Exercise 4, but this time use the EXISTS operator in your query.
6. List the names of any pair of boats that have the same type. For example, one pair would be *Anderson II* and *Escape*, because the boat type for both boats is Sprite 4000. The first name listed should be the major sort key and the second name should be the minor sort key.
7. List the boat name, owner number, owner last name, and owner first name for each boat in marina 1.
8. Repeat Exercise 7, but this time only list boats in 40-foot slips.
9. List the marina number, slip number, and boat name for boats whose owners live in Glander Bay or whose type is Sprite 4000.
10. List the marina number, slip number, and boat name for boats whose owners live in Glander Bay and whose type is Sprite 4000.
11. List the marina number, slip number, and boat name for boats whose owners live in Glander Bay but whose type is not Sprite 4000.
12. Find the service ID and slip ID for each service request whose estimated hours is greater than the number of estimated hours of at least one service request on which the category number is 3.
13. Find the service ID and slip ID for each service request whose estimated hours is greater than the number of estimated hours on every service request on which the category number is 3.
14. List the slip ID, boat name, owner number, service ID, number of estimated hours, and number of spent hours for each service request on which the category number is 2.
15. Repeat Exercise 14, but this time be sure each slip is included regardless of whether the boat in the slip currently has any service requests for category 2.

CHAPTER 5—MULTIPLE-TABLE QUERIES

1. Indicate in the SELECT clause all columns to display, list in the FROM clause all tables to join, and then include in the WHERE clause any conditions requiring values in matching columns to be equal.
3. IN and EXISTS
5. An alias is an alternate name for a table. To specify an alias in SQL, follow the name of the table with the name of the alias. You use the alias just like a table name throughout the SQL command.
7. Use the UNION, INTERSECT, and MINUS operators to create a union, intersection, and difference of two tables. To perform any of these operations, the tables must be union compatible.

9. When the ALL operator precedes a subquery, the condition is true only if it satisfies all values produced by the subquery.
11. In an inner join, only matching rows from both tables are included. You can use the INNER JOIN clause to perform an inner join.
13. In a right outer join, all rows from the table on the right will be included regardless of whether they match rows from the table on the left. Rows from the table on the left will be included only if they match. You can use the RIGHT JOIN clause to perform a right outer join.
15. Answers will vary. Answers should note that an equi-join is similar to an inner join except that both matching columns appear in the results. A natural join is the same as the inner join discussed in Chapter 5. A cross join is the same as a Cartesian product.