Creating SQL Queries in Microsoft SQL Server

You will begin by reading the “Introduction” from page 71 in the Pratt and Last textbook as well as the section “Table Creation” on pages 72 through 73. Then continue below with the section on “Queries” in this tutorial.

QUERIES

A query is a “question” that asks the DBMS (database management system) to retrieve a subset of records and columns from one or more tables in a database. In SQL Server, SQL SELECT queries are created and saved as objects within the “View” window as CREATE VIEW statements.

Start Visual Studio and open the Web Site with the database. To reach the “View” window in the “Server Explorer” window click the “expand” [+] arrow for the database to view its objects. Right-click on the “Views” group and from the short-cut menu click the “Add New View” command.

New queries, called Views in SQL Server, are created in a statement that has two parts:

1. The CREATE VIEW statement (e.g. “CREATE VIEW viewName AS”) which executes when the <Update> button is clicked and actually create the query
2. The SQL SELECT statement (e.g. “SELECT ...”) that follows is referred to as the defining query and is the actual query that retrieves a “relation” (a subset of the database)

When the “View” window first opens, you will see a template, a starting point for the new statement. The “squiggly” blue and red lines under parts of the text indicate that the user must “fill in” elements in the statement before it will be a valid SQL statement.
For each new query the first thing to do is to give the View a name. This is accomplished by changing the name [View]. In the example below the View has been changed to “Test”. For the time being the only other element to be changed will be the table name in the SELECT statement which has been updated to “Rep”.

After completing the query, the user will create the View. This is done the same way that a Table is created or updated, that is by clicking the <Update> button.

The “Preview Database Updates” window opens. As long as there are no errors in the statement, it indicates that a new View will be created. Click the <Update Database> button.
The new View has been created. To see it the user returns to the “Server Explorer” window. If necessary click the “expand” [+] arrow in front of the “Views” group to see the list of View objects. Also if necessary right-click on “Views” and select “Refresh” (just like you do for Table objects) from the short-cut menu to see the new View object.

Now the user will want to see the query results. To do this right-click on the name of the View and select “Show Results” from the short-cut menu (see next page) which executes the SQL SELECT statement.
After a few moments once the DBMS has executed the query, the results are displayed as in the image below:

SIMPLE RETRIEVAL

The basic form of a SQL retrieval command is SELECT-FROM-WHERE. After the word SELECT, you list the fields you want to display in the query results. This portion of the command is called the SELECT clause. The fields will appear in the query results in the order in which they are listed in the SELECT clause. After the word FROM, you list the table or tables that contain the data to display in the query results. This portion of the command is called the FROM clause. Finally, after the word WHERE, you list any conditions that you want to apply to the data you want to retrieve, such as indicating that the credit limit must be $10,000. This portion of the command, which is optional, is called the WHERE clause.

There are no special formatting rules in SQL— the examples in this text include the SELECT, FROM, and WHERE clauses on separate lines to make the commands more readable. In addition, this
text uses a common style in which words that are part of the SQL language, called **reserved words**, appear in all uppercase letters. All other words in commands appear in a combination of uppercase and lowercase letters.

**EXAMPLE 2**

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

Because you want to list all customers, you won’t need to use the WHERE clause— you don’t need to put any restrictions on the data to retrieve.

The CREATE VIEW statement and SQL SELECT query is:

```
CREATE VIEW [dbo].[Example2] 
AS

SELECT   CustomerNum, CustomerName, Balance 
FROM      Customer
```

The image below shows the query to select the number, name, and balance of all customers, using the SQL implementation in SQL Server. The name of the View has been changed to “Example2” and the SQL SELECT statement follows the reserved word AS.

Once the query is complete, click the `<Update>` button to create the View object. Then return to the “Server Explorer” window and refresh the list of Views if necessary to see the new object. Finally right-click on the name of the new View and select “Show Results” from the short-cut menu. The results should be similar to those in the image below.
You could use the same approach shown in Example 2 by listing each field in the Part table in the SELECT clause. However, there is a shortcut. Instead of listing all the field names in the SELECT clause, you can use the * symbol. When used after the word `SELECT`, the * symbol indicates that you want to include all fields in the query results in the order in which you described them to the DBMS when you created the table. To include all the fields in the query results, but in a different order, you would type the names of the fields in the order in which you want them to appear. In this case, assuming the default order is appropriate. The query results appears in the image below.

The CREATE VIEW statement and SQL SELECT query is:

```sql
CREATE VIEW [dbo].[Example3]
AS
SELECT *
FROM Part
```
The query results above are displayed in the output window when the “Show Results” command is selected from the “Views” short-cut menu.

When the records that you want to display in a query’s results must satisfy a condition, you use the keyword WHERE. Conditions that data must satisfy are also called criteria. (A single condition is called a criterion.) The following example illustrates the use of the WHERE clause to select data.

**EXAMPLE 4**
List the name of every customer with a $10,000 credit limit.

You include the following condition in the WHERE clause to restrict the query results to only those customers with a credit limit of $10,000.

```
WHERE CreditLimit = 10000
```

Notice that you do not type commas or dollar signs in numbers. The query design appears below.

The CREATE VIEW statement and SQL SELECT query is:

```
CREATE VIEW [dbo].[Example4]
AS

SELECT CustomerName
FROM Customer
WHERE CreditLimit = 10000
```
The WHERE clause shown above includes a simple condition. A **simple condition** includes the field name, a **comparison operator** (also called a **relational operator**), and either another field name or a value, such as `CreditLimit = 10000` or `CreditLimit > Balance`. The table below (next page) lists the comparison operators that you can use in SQL commands. Notice that there are two versions of the “not equal to” operator: `<>` and `!=`. You must use the correct one for your version of SQL. If you use the wrong one, your system will generate an error, in which case, you’ll know to use the other version. SQL Server uses `<>`.

<table>
<thead>
<tr>
<th>Comparison Operator</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>+</code></td>
<td>Equal to</td>
</tr>
<tr>
<td><code>&lt;</code></td>
<td>Less than</td>
</tr>
<tr>
<td><code>&gt;</code></td>
<td>Greater than</td>
</tr>
<tr>
<td><code>&lt;=</code></td>
<td>Less than or equal to</td>
</tr>
<tr>
<td><code>&gt;=</code></td>
<td>Greater than or equal to</td>
</tr>
<tr>
<td><code>&lt;&gt;</code></td>
<td>Not equal to (used by most implementations of SQL)</td>
</tr>
<tr>
<td><code>!=</code></td>
<td>Not equal to (used by some implementations of SQL)</td>
</tr>
</tbody>
</table>

**Comparison (relational) operators used in SQL commands**

The query results above are displayed in the output window when the “Show Results” command is selected from the “Views” short-cut menu.

In Example 4, the WHERE clause compared a numeric field (CreditLimit) to a number (10000). When a query involves a character field, such as CustomerName, you must enclose the value to which the field is being compared in single quotation marks, as illustrated in Examples 5 and 6.

**EXAMPLE 5**

Find the name of customer 148.

The query design appears below. Because CustomerNum is an **varchar** field, the value 148 is enclosed in single quotation marks.

The CREATE VIEW statement and SQL SELECT query is:
CREATE VIEW [dbo].[Example5] AS
SELECT CustomerName
FROM Customer
WHERE CustomerNum = '148'

The query results above are displayed in the output window when the “Show Results” command is selected from the “Views” short-cut menu. Only a single record appears in the query results because the CustomerNum field is the primary key for the Customer table and there can be only one customer with the number 148.

**EXAMPLE 6**
Find the customer name for every customer located in the city of Grove.

The query design appears below.

The CREATE VIEW statement and SQL SELECT query is:

```
CREATE VIEW [dbo].[Example6] AS
SELECT CustomerName
FROM Customer
WHERE City = 'Grove'
```
The query results above are displayed in the output window when the “Show Results” command is selected from the “Views” short-cut menu. Because more than one customer is located in Grove, there are multiple records in the query results.

You can also use dates in conditions. The format for using dates in queries varies slightly from one implementation of SQL to another. In Access, you place number signs around the date (for example, # 11/15/2013#). In other programs, you enter the day of the month, a hyphen, the three-character abbreviation for the month, a hyphen, and the year, all enclosed in single quotation marks (for example, ’15-NOV-2013’). SQL Server uses a combination of the two formats placing single quotation marks around the ’mm/dd/yyyy’ date (for example, ’11/15/2013’).

**EXAMPLE 7**

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

The query design appears below. Notice that the condition in the WHERE clause compares the contents of two fields.

The CREATE VIEW statement and SQL SELECT query is:

```
CREATE VIEW [dbo].[Example7] AS
SELECT CustomerNum, CustomerName, CreditLimit, Balance
FROM Customer
WHERE CreditLimit > Balance
```
The query results above are displayed in the output window when the “Show Results” command is selected from the “Views” short-cut menu.

**COMPOUND CONDITIONS**

The conditions you’ve seen so far are called simple conditions. The following examples require compound conditions. A **compound condition** is formed by connecting two or more simple conditions using one or both of the following operators: **AND** and **OR**. You can also precede a single condition with the **NOT** operator to negate a condition.

When you connect simple conditions using the **AND** operator, all the simple conditions must be true for the compound condition to be true. When you connect simple conditions using the **OR** operator, the compound condition will be true whenever any of the simple conditions are true. Preceding a condition by the **NOT** operator reverses the truth or falsity of the original condition. That is, 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 8**

List the descriptions of all parts that are located in warehouse 3 and for which there are more than 20 units on hand.

In this example, you want to list those parts for which both the warehouse number is equal to 3 and the number of units on hand is greater than 20. Thus, you form a compound condition using the **AND** operator, as shown below.

The **CREATE VIEW** statement and SQL SELECT query is:

**CREATE VIEW** [dbo].[Example8]
AS

SELECT Description
FROM Part
WHERE Warehouse = 3
AND OnHand > 20

The query results above are displayed in the output window when the “Show Results” command is selected from the “Views” short-cut menu.

EXAMPLE 9
List the descriptions of all parts that are located in warehouse 3 or for which there are more than 20 units on hand or both.

As you would expect, you form compound conditions with the OR operator similar to the way you use the AND operator. The compound condition shown in the image below uses the OR operator instead of the AND operator.

The CREATE VIEW statement and SQL SELECT query is:

CREATE VIEW [dbo].[Example9]
AS

SELECT Description
FROM Part
WHERE Warehouse = 3
OR OnHand > 20
The query results above are displayed in the output window when the “Show Results” command is selected from the “Views” short-cut menu.

**EXAMPLE 10**
List the descriptions of all parts that are *not* in warehouse 3.

For this example, you could use a simple condition and the “not equal to” operator (<>). As an alternative, you could use the “equals” operator (=) in the condition, but precede the entire condition with the NOT operator, as shown below.

The CREATE VIEW statement and SQL SELECT query is:

```
CREATE VIEW [dbo].[Example10]
AS

SELECT    Description
FROM       Part
WHERE      NOT Warehouse = '3'
```
The query results above are displayed in the output window when the “Show Results” command is selected from the “Views” short-cut menu.

**EXAMPLE 11**
List the number, name, and balance of all customers with balances greater than or equal to $1,000 and less than or equal to $5,000.

You could use a WHERE clause and the AND operator (Balance>= 1000 AND Balance<= 5000). An alternative to this approach uses the BETWEEN operator, as shown below.

The CREATE VIEW statement and SQL SELECT query is:

```sql
CREATE VIEW [dbo].[Example11] AS
SELECT CustomerNum, CustomerName, Balance
FROM Customer
WHERE Balance BETWEEN 1000 AND 5000
```
The query results above are displayed in the output window when the “Show Results” command is selected from the “Views” short-cut menu.

The BETWEEN operator is not an essential feature of SQL; you can use the AND operator to obtain the same results. Using the BETWEEN operator, however, does make certain SELECT commands easier to construct.

**COMPUTED FIELDS**

Similar to QBE, you can include fields in queries that are not in the database but whose values you can compute from existing database fields. A field whose values you derive from existing fields is called a computed field or calculated field. Computed fields can involve addition (+), subtraction (−), multiplication (*), or division. The query in Example 12, for example, uses subtraction.

**EXAMPLE 12**

List the number, name, and available credit for all customers.

There is no field in the database that stores available credit, but you can compute it using two fields that are present in the database: CreditLimit and Balance. The query design shown in the image below creates a new field named AvailableCredit, which is computed by subtracting the value in the Balance field from the value in the CreditLimit field (AvailableCredit = CreditLimit − Balance). By using the word AS after the computation, followed by AvailableCredit, you can assign a column name (also known as an alias) to the computed field.

The CREATE VIEW statement and SQL SELECT query is:

```
CREATE VIEW [dbo].[Example12]
AS

SELECT    CustomerNum, CustomerName, CreditLimit - Balance AS AvailableCredit
FROM       Customer
```
The query results above are displayed in the output window when the “Show Results” command is selected from the “Views” short-cut menu. The column heading “AvailableCredit” for the computed field is the name (alias) that you specified in the SELECT clause.

Computations are not limited to values in number fields. You can combine values in character fields as well. For example, in SQL Server you can combine the values in the FirstName and LastName fields into a single computed field by using the + operator (Access uses the & operator). The expression would be FirstName + ’ ’ + LastName, which places a space between the first name and the last name. The formal term is that you are concatenating the FirstName and LastName fields. In SQL Server (and MySQL as well), you use the CONCAT function to concatenate fields. To concatenate FirstName and LastName, for example, the expression would be CONCAT( FirstName, LastName).

**EXAMPLE 13**

List the number, name, and available credit for all customers with credit limits that exceed their balances.

The only difference between Examples 12 and 13 is that Example 13 includes a condition, as shown below.
The CREATE VIEW statement and SQL SELECT query is:

```
CREATE VIEW [dbo].[Example13]  
AS  
SELECT  
    CustomerNum, CustomerName, CreditLimit - Balance AS AvailableCredit  
FROM  
    Customer  
WHERE  
    CreditLimit > Balance
```

The query results above are displayed in the output window when the “Show Results” command is selected from the “Views” short-cut menu.

**USING SPECIAL OPERATORS (LIKE AND IN)**

In most cases, your conditions will involve exact matches, such as finding all customers located in the city of Sheldon. In some cases, however, exact matches will not work. For example, you might know only that the desired value contains a certain collection of characters. In such cases, you use the LIKE operator with a wildcard, as shown in Example 14.

**EXAMPLE 14**

List the number, name, and complete address of every customer located on a street that contains the letters *Oxford*.

All you know is that the addresses that you want contain a certain collection of characters (Oxford) somewhere in the Street field, but you don’t know where. In SQL Server (and MySQL as well), the percent sign (%) is used as a wildcard to represent any collection of characters. (In Access SQL, the asterisk (*) is used as a wild-card to represent any collection of characters.) To use a wildcard, include the LIKE operator in the WHERE clause. The query design shown below will retrieve information for
every customer whose street contains some collection of characters followed by the letters Oxford, followed potentially by some additional characters.

The CREATE VIEW statement and SQL SELECT query is:

```sql
CREATE VIEW [dbo].[Example14]
AS
SELECT CustomerNum, CustomerName, Street, City, State, Zip
FROM Customer
WHERE Street LIKE '%%Oxford%%'
```

The query results above are displayed in the output window when the “Show Results” command is selected from the “Views” short-cut menu.

Another wildcard in SQL Server (an also MySQL) 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 when used in a WHERE clause, retrieves records that include the words Tim, Tom, or T3m, for example. Many Access version of SQL uses a question mark (?) instead of the underscore to represent any individual character.

**NOTE**

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

Another operator, IN, provides a concise way of phrasing certain conditions.

**EXAMPLE 15**

List the number, name, street, and credit limit for every customer with a credit limit of $7,500, $10,000, or $15,000.

In this query, you can use the SQL IN operator to determine whether a credit limit is $7,500, $10,000, or $15,000. You can obtain the same result by using the condition WHERE CreditLimit = 7500 OR CreditLimit = 10000 OR CreditLimit = 15000. The approach shown below is simpler, however— the
IN clause contains the collection of values 7500, 10000, and 15000 listed inside (parentheses). The condition is true for rows in which the value in the CreditLimit column is in this collection of values.

The CREATE VIEW statement and SQL SELECT query is:

```
CREATE VIEW [dbo].[Example15]
AS
SELECT CustomerNum, CustomerName, Street, CreditLimit
FROM Customer
WHERE CreditLimit IN (7500,10000,15000)
```

The query results above are displayed in the output window when the “Show Results” command is selected from the “Views” short-cut menu.

**SORTING**

Recall that the order of rows in a table is considered to be immaterial. From a practical standpoint, this means that when you query a relational database, there are no guarantees concerning the order in which the results will be displayed. The results might appear in the order in which the data was originally entered, but even this is not certain. Thus, if the order in which the data is displayed is important, you should specifically request that the results be displayed in a desired order. In SQL, you sort data using the ORDER BY clause.

**EXAMPLE 16**
List the number, name, street, and credit limit of all customers. Order (sort) the customers by name.
The field on which to sort data is called a **sort key**. To sort the output, you include the words `ORDER BY` in the SQL query, followed by the sort key field, as shown below. Additionally, the SQL Server implementation of SQL requires a count of the number of records to be retrieved by a SELECT statement that includes an `ORDER BY` clause. For our examples in this text, this will be accomplished by including the clause `TOP 100` (meaning the first 100 records) after the SELECT command in all statements that incorporate a sort query. (Of course, we do not have close to 100 records in any of our tables so this effectively always will retrieve all records.)

The CREATE VIEW statement and SQL SELECT query is:

```sql
CREATE VIEW [dbo].[Example16] 
AS 
SELECT TOP 100 CustomerNum, CustomerName, Street, CreditLimit 
FROM Customer 
ORDER BY CustomerName 
```

The query results above are displayed in the output window when the “Show Results” command is selected from the “Views” short-cut menu. Notice they are sort alphabetically by `CustomerName` and not by `CustomerNum` (the primary key).

**EXAMPLE 17**

List the number, name, street, and credit limit of all customers. Order the customers by name within descending credit limit. (In other words, 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.)
When you need to sort data on two fields, the more important sort key is called the major sort key (also referred to as the primary sort key) and the less important sort key is called the minor sort key (also referred to as the secondary sort key). In this case, because you need to sort the output by name within credit limit, the CreditLimit field is the major sort key and the CustomerName field is the minor sort key. If there are two sort keys, as in the image below, the major sort key will be listed first. You can specify to sort the output in descending (high-to-low) in descending (high-to-low) order by following the sort key with the word **DESC**, as shown below.

The CREATE VIEW statement and SQL SELECT query is:

```sql
CREATE VIEW [dbo].[Example17] AS
SELECT TOP 100 CustomerNum, CustomerName, Street, CreditLimit
FROM Customer
ORDER BY CreditLimit DESC, CustomerName
```

The query results above are displayed in the output window when the “Show Results” command is selected from the “Views” short-cut menu.

**BUILT-IN FUNCTIONS**

SQL has built-in functions (also called aggregate functions) to calculate the number of entries, the sum or average of all the entries in a given column, and the largest or smallest values in a given column. In SQL, these functions are called **COUNT**, **SUM**, **AVG**, **MAX**, and **MIN**, respectively.
EXAMPLE 18
How many parts are in item class HW?

In this query, you need to count the number of rows in the query results that have the value HW in the Class field. You could count the number of part numbers in the query results or the number of descriptions or the number of entries in any other field. It doesn’t matter which column you choose because all columns will yield the correct answer. Rather than requiring you to pick a column arbitrarily, some versions of SQL allow you to use the * symbol to select any column.

As with computed fields, you use the reserved word AS to assign names (aliases) to these computations, as shown below. This is an option in some dialects of SQL but it is a requirement in SQL Server SELECT statements that use built-in functions.

In SQL versions that support the * symbol, you could use the query design shown below.

The CREATE VIEW statement and SQL SELECT query is:

```
CREATE VIEW [dbo].[Example18]
AS
SELECT COUNT(*) AS CountOfHardwareParts
FROM Part
WHERE Class = 'HW'
```

The query results above are displayed in the output window when the “Show Results” command is selected from the “Views” short-cut menu.

If your implementation of SQL doesn’t permit the use of the * symbol, you could write the query as follows:

```
SELECT COUNT(PartNum) AS CountOfHardwareParts
FROM Part
WHERE Class = 'HW'
```
EXAMPLE 19
Find the number of customers and the total of their balances.

(Example 19 has been eliminated from this text since it is identical to Example 20 which performs the same operations but with a column name (alias). Remember that aliases for built-in functions are required in SQL Server.)

EXAMPLE 20
Find the total number of customers and the total of their balances. Change the column names for the number of customers and the total of their balances to CustomerCount and BalanceTotal, respectively.

There are two differences between COUNT and SUM—other than the obvious fact that they are computing different statistics. In the case of SUM, you must specify the field for which you want a total and the field must be numeric. (How could you calculate a sum of names or addresses?) The query design is shown below.

The CREATE VIEW statement and SQL SELECT query is:

```
CREATE VIEW [dbo].[Example20]
AS
SELECT COUNT(*) AS CustomerCount, SUM(Balance) AS BalanceTotal
FROM Customer
```

The query results above are displayed in the output window when the “Show Results” command is selected from the “Views” short-cut menu.

SUBQUERIES

In some cases, it is useful to obtain the results you want in two stages. You can do so by placing one query inside another. The inner query is called a subquery and is evaluated first. After the subquery has been evaluated, the outer query can be evaluated. Example 21 illustrates the process.

EXAMPLE 21
List the order number for each order that contains an order line for a part located in warehouse 3.
The query design appears in the image below.

The CREATE VIEW statement and SQL SELECT query is:

```sql
CREATE VIEW [dbo].[Example21] AS
SELECT  OrderNum
FROM    OrderLine
WHERE   PartNum IN
        (SELECT  PartNum
         FROM    Part
         WHERE   Warehouse = '3')
```

The query results above are displayed in the output window when the “Show Results” command is selected from the “Views” short-cut menu. The subquery finds all the part numbers in the Part table with a warehouse number of 3. The subquery is evaluated first, producing a list of part numbers. After the subquery has been evaluated, the outer query is evaluated. Order numbers in the results appear in any row in the OrderLine table for which the part number in the row is in the subquery results.

GROUPING

Recall from Chapter 2 that grouping means creating groups of records that share some common characteristic. When grouping customers by sales rep number, for example, the customers of sales rep 20 would form one group, the customers of sales rep 35 would form a second group, and the customers of sales rep 65 would form a third group.

In Example 22, you need to group customers by rep number to perform the necessary calculations.
For each sales rep, list the rep number, the number of customers assigned to the rep, and the average balance of the rep’s customers. Group the records by rep number and order the records by rep number.

This type of query requires grouping by rep number to make the correct calculations for each group. To indicate grouping in SQL, you use the `GROUP BY` clause, as shown below. It is important to note that the `GROUP BY` clause does not mean that the query results will be sorted. To display the query results in a particular order, you must use the `ORDER BY` clause. The query design below uses the `ORDER BY` clause to sort the query results by rep number.

The CREATE VIEW statement and SQL SELECT query is:

```
CREATE VIEW [dbo].[Example22]
AS
SELECT TOP 100 RepNum, COUNT(*) AS CountOfReps, AVG(Balance) AS AverageBalance
FROM Customer
GROUP BY RepNum
ORDER BY RepNum
```

The query results above are displayed in the output window when the “Show Results” command is selected from the “Views” short-cut menu.

When rows are grouped, one line of output is produced for each group. Only statistics calculated for the group or fields whose values are the same for all rows in a group can be displayed in the grouped results.

**Q & A**

**Question:** Why is it appropriate to display the rep number?

**Answer:** Because the output is grouped by rep number, 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? Why or why not?
Answer: No, because the customer number will vary from one row in a group to another. (SQL could not determine which customer number to display for the group.)

EXAMPLE 23
For each sales rep with fewer than four customers, list the rep number, the number of customers assigned to the rep, and the average balance of the rep’s customers. Rename the count of the number of customers and the average of the balances to NumCustomers and AverageBalance, respectively. Order the groups by rep number.

Examples 22 and 23 are similar, but there is an important difference. There is a restriction to display the calculations for only those reps having fewer than four customers. In other words, you want to display only those groups for which COUNT(*) is less than four. This restriction does not apply to individual rows, but to groups. Because the WHERE clause applies only to rows, it is not the appropriate clause to accomplish the kind of selection you need. Fortunately, the HAVING clause is to groups what the WHERE clause is to rows, as shown below.

The CREATE VIEW statement and SQL SELECT query is:

```
CREATE VIEW [dbo].[Example23] AS
SELECT TOP 100 RepNum, COUNT(*) AS NumCustomers, AVG(Balance) AS AverageBalance
FROM Customer
GROUP BY RepNum
HAVING COUNT(*) < 4
ORDER BY RepNum
```

The query results above are displayed in the output window when the “Show Results” command is selected from the “Views” short-cut menu. In this case, the row created for a group will be displayed only when the count of the number of records in the group is less than four.
You can include both a WHERE clause and a HAVING clause in the same query design, as shown below.

The CREATE VIEW statement and SQL SELECT query is:

```sql
CREATE VIEW [dbo].[View1]
AS
SELECT TOP 100 RepNum, COUNT(*) AS NumCustomers, AVG(Balance) AS AverageBalance
FROM Customer
WHERE CreditLimit < 10000
GROUP BY RepNum
HAVING COUNT(*) < 3
ORDER BY RepNum
```

In this case, the condition in the WHERE clause restricts the rows from the Customer table to those rows in which the credit limit is less than $10,000. These rows are grouped by rep number. The HAVING clause then restricts the groups to those for which the count of the rows in the group is less than three.

**JOINING TABLES**

Many queries require data from more than one table. As with QBE and relational algebra, it is necessary to be able to join tables so you can find rows in two or more tables that have identical values in matching fields. In SQL, this is accomplished by entering the appropriate conditions in the WHERE clause. (Appendix B at the end of this text includes information about an alternative way of joining tables in SQL that uses the FROM clause.)

**EXAMPLE 24**

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. Order the records by customer number.
Because the numbers and names of customers are in the Customer table and the numbers and names of sales reps are in the Rep table, you need to include both tables in your SQL query. To join the tables, you’ll construct the SQL command as follows:

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

As in relational algebra, it is often necessary to qualify a field name to specify the particular field you are referencing. To qualify a field name, precede the name of the field with the name of the table, followed by a period. For example, the RepNum field in the Rep table is written as Rep.RepNum and the RepNum field in the Customer table is written as Customer.RepNum. The query design appears in the image below.

The CREATE VIEW statement and SQL SELECT query is:

CREATE VIEW [dbo].[Example24]
AS

SELECT TOP 100 CustomerNum, CustomerName, Rep.RepNum, LastName, FirstName
FROM Customer, Rep
ORDER BY CustomerNum
The query results above are displayed in the output window when the “Show Results” command is selected from the “Views” short-cut menu.

When there is potential ambiguity in listing field names, you must qualify the fields involved. It is permissible to qualify other fields as well, even if there is no possible confusion. Some people prefer to qualify all fields, which is certainly not a bad approach. In this text, however, you will qualify fields only when it is necessary to do so.

**EXAMPLE 25**

List the number and name of each customer whose credit limit is $10,000 together with the number, last name, and first name of the sales rep who represents the customer. Order the records by customer number.

In Example 25, the condition in the WHERE clause serves only to relate a customer to a sales rep. Although relating a customer to a sales rep is essential in this example as well, you also need to restrict the output to only those customers whose credit limit is $10,000. You can accomplish this goal by using the AND operator to create a compound condition, as shown below.

The CREATE VIEW statement and SQL SELECT query is:

```
CREATE VIEW [dbo].[Example25]
AS
SELECT CustomerNum, CustomerName, Rep.RepNum, LastName, FirstName
FROM Customer, Rep
AND CreditLimit = 10000
ORDER BY CustomerNum
```

The query results above are displayed in the output window when the “Show Results” command is selected from the “Views” short-cut menu.
It is possible to join more than two tables, as illustrated in Example 26. For each pair of tables to join, you must include a condition indicating how the tables are related.

**EXAMPLE 26**
For every order, list the order number, order date, customer number, and customer name. In addition, for each order line within the order, list the part number, description, number ordered, and quoted price. Order the records by order number.

The order number and date are stored in the Orders table. The customer number and name are stored in the Customer table. The part number and description are stored in the Part table. The number ordered and quoted price are stored in the OrderLine table. Thus, you need to join four tables: Orders, Customer, Part, and OrderLine. 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, as shown below. The first condition relates an order to a customer, using the common CustomerNum columns. The second condition relates the order to an order line, using the common OrderNum columns. The final condition relates the order line to a part, using the common PartNum columns.

The CREATE VIEW statement and SQL SELECT query is:

```
CREATE VIEW [dbo].[Example26]
AS
SELECT TOP 100 Orders.OrderNum, OrderDate, Customer.CustomerNum, CustomerName, Part.PartNum, Description, NumOrdered, QuotedPrice
FROM Orders, Customer, OrderLine, Part
WHERE Customer.CustomerNum = Orders.CustomerNum
AND Orders.OrderNum = OrderLine.OrderNum
AND OrderLine.PartNum = Part.PartNum
ORDER BY Orders.OrderNum
```
The query results above are displayed in the output window when the “Show Results” command is selected from the “Views” short-cut menu. This query is more complex than many of the previous ones. 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, you will find that the query in Example 26 isn’t all 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 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 must also 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 sales rep name, you wouldn’t include any columns from the Customer table in the SELECT clause. The Customer table is still required in the FROM clause, however, because you must include columns 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. When there are other conditions, include them in the WHERE clause and connect them to the other conditions with the AND operator.

**UNION**

Recall from Chapter 2 that the union of two tables is a table containing all rows that are in the first table, the second table, or both tables. The two tables involved in a union must have the same structure, or be union compatible; in other words, they must have the same number of fields and their
corresponding fields must have the same data types. If, for example, the first field in one table contains customer numbers, the first field in the other table also must contain customer numbers.

### EXAMPLE 27
List the number and name of all customers that are represented by sales rep 35 or that currently have orders on file or both.

Because the two criteria are so different, you cannot use a simple OR criterion. Instead, you can create a table containing the number and name of all customers that are represented by sales rep 35 by selecting customer numbers and names from the Customer table in which the sales rep number is 35. You can then create another table containing the number and name of every customer that currently has orders on file by joining the Customer and Orders tables. The two tables created by this process have the same structure—fields named CustomerNum and CustomerName. Because the tables are union compatible, it is possible to take the union of these two tables, which is the appropriate operation for this example, as shown below.

The CREATE VIEW statement and SQL SELECT query is:

```sql
CREATE VIEW [dbo].[Example27] AS
SELECT CustomerNum, CustomerName
FROM Customer
WHERE RepNum = '25'
UNION
SELECT Customer.CustomerNum, CustomerName
FROM Customer, Orders
WHERE Customer.CustomerNum = Orders.CustomerNum
```
The query results above are displayed in the output window when the “Show Results” command is selected from the “Views” short-cut menu. If a SQL implementation truly supports the union operation, it will remove any duplicate rows. For instance, any customers that are represented by sales rep 35 and that currently have orders on file will not appear twice in the query results. Some SQL implementations have a union operation but will not remove duplicate values.

UPDATING TABLES

There are more capabilities for SQL, than simply retrieving data from a database and creating tables. SQL has several other capabilities, including the ability to update a database as demonstrated in the following examples.

**NOTE**

If you plan to work through the examples in this section using SQL Server, you should use a copy of the original Premiere Products database because the version of the database used in subsequent chapters does not include these changes.

**Update queries** usually are one-time events and as such are not objects to be saved as are “Views” in “Server Explorer.” Instead these operations are entered into the SQL Server’s “Query” window which also is reached by right-clicking on the “Views” group. However instead of selecting “Add New View,” click the command “New Query” from the short-cut menu.

Any SQL command, CREATE TABLE, SELECT, INSERT, UPDATE, DELETE, etc., can be entered into the “Query” window. You run it once and then are done with it. These queries are not saved in SQL Server.
To execute a query in the “Query” window, click the <Execute> button (there is no <Update> button available) as in the image below.

![Image of Premier.sln - Microsoft Visual Studio](image)

The processing results of the SQL operation that was performed (or an error message) will appear in the “T-SQL” window which opens at the bottom half of the SQL Server IDE. You can close the “Query” window when you are done with it but you cannot save anything.

In general the “Query” window is not intended for SQL SELECT commands as in the example above. Rather it is used more frequently for other operations that involve updating the database, either its structure or data within the tables.

**EXAMPLE 28**
Change the street address of customer 524 to 1445 Rivard.

You can use the SQL UPDATE command to make changes to existing data. After the word **UPDATE**, you indicate the table to be updated. After the word **SET**, you indicate the field (or fields) to be changed followed by an equal sign and the new value. Finally you can include a condition in a **WHERE** clause in which case only the record(s) that satisfy the condition will be changed.

The format for an UPDATE statement is as follows:

- **UPDATE**  \( \text{tableName} \)
- **SET**  \( \text{firstColumnName} = \text{firstValue} [, \text{secondColumnName} = \text{secondValue}, \text{etc.}] \)
- **WHERE**  \( \text{booleanExpression} \)
The UPDATE query statement is:

```
UPDATE Customer
SET Street = '1445 Rivard'
WHERE CustomerNum = '524'
```

When you click the <Execute> button the “T-SQL” window opens and indicates the number of records the UPDATE command will affect. In this case, you would update only one record because the WHERE clause selects customer 524.

You can select “Show Table Data” for the Customer table to verify that this has occurred.

```
`1 row(s) affected`
```

EXAMPLE 29
Add a new sales rep to a table. Her number is 16; her name is Sharon Rands; and her address is 826 Raymond, Altonville, FL, 32543. She has not yet earned any commission, but her commission rate is 5% (0.05).

To add new data to a table, you use the INSERT command. After the words INSERT INTO, you list the name of the table, followed by the word VALUES. Then you list the values in parentheses for each of the columns in the table. Character values (e.g. VARCHAR data types) must be enclosed within single quotation marks.

The format for a SELECT/INTO statement is as follows:

```
INSERT INTO <tableName> VALUES (<list of values that match the columns in the table>)
```

The INSERT query statement is:

```
INSERT INTO Rep
VALUES (16, 'Rands', 'Sharon', '826 Raymond', 'Altonville', 'FL', '32543', 0.00, 0.05)
```
When you click the `<Execute>` button the "T-SQL" window opens and indicates the number of records the INSERT command will append to the table. In this case you would add one record to the Rep table.

You can select "Show Table Data" for the Rep table to verify that this has occurred.

**EXAMPLE 30**
Delete any row in the OrderLine table in which the part number is BV06.

To delete data from the database, you use the **DELETE** command, which consists of the word **DELETE** followed by a FROM clause indicating the table. Use a WHERE clause to specify a condition to select the records to delete. If you omit the condition for selecting records to delete, when you run the query, it will delete all records from the table.

The format for a DELETE statement is as follows:

```
DELETE
FROM   tableName
WHERE  booleanExpression
```

The DELETE query statement is:

```
DELETE
FROM   OrderLine
WHERE  ProductNum = 'BV06'
```
When you click the <Execute> button the “T-SQL” window opens and indicates the number of records the DELETE command will delete. In this case, you would delete only one record because the WHERE clause selects part number BV06.

You can select “Show Table Data” for the OrderLine table to verify that this has occurred.

CREATING A TABLE FROM A QUERY

You can save the results of a SQL Server SELECT query as a new table by including the INTO clause in the query, as illustrated in Example 31. The format for a SELECT/INTO statement is as follows:

```
SELECT columnList
INTO newTableName
FROM existingTable(s)
(plus any additional SELECT statement processing including WHERE, ORDER BY, etc.)
```

**EXAMPLE 31**

Create a new table name SmallCust consisting of all fields from the Customer table and those rows in which the credit limit is less than or equal to $7,500.

To create the SmallCust table, from Server Explorer open a new “Query” window. Create a query to select all fields from the Customer table, include a WHERE clause to restrict the rows to those in which CreditLimit <= 7500, and include an INTO clause. The INTO clause precedes the FROM clause and consists of the word **INTO** followed by the name of the table to be created.

The SELECT INTO query statement is:

```
SELECT * INTO SmallCust FROM Customer WHERE CreditLimit <= 7500
```
When you click the `<Execute>` button the “T-SQL” window opens and indicates the number of records the INTO clause will paste into the new table, in this case six rows to the SmallCust table.

![SQL query execution](image)

After you execute this query, you can use the SmallCust table (which can be found in the “Server Explorer” window as per the image below) just like any other table you created using the CREATE TABLE command.
Once you have completed this tutorial continue by reviewing the “Summary of SQL Commands” on pages 103 through 110 in the Pratt and Last textbook.