**Introducing T-SQL**

T-SQL, or Transact-SQL, is the dialect of the Structured Query Language (SQL) incorporated in SQL

Server. To work effectively as a SQL Server developer, you must have a strong grasp of T-SQL.

Fortunately, T-SQL is easy to learn. When retrieving data, you simply build a SELECT statement.

SELECT statements are composed of clauses that determine the specifics of how the data is selected.

When they're executed, SELECT statements select rows of data and return them as a recordset.

**By the Way**

In the examples that follow, keywords appear in uppercase. Values that you supply

appear italicized. Optional parts of the statement appear in square brackets. Curly braces,

combined with vertical bars, indicate a choice. Finally, ellipses are used to indicate a

repeating sequence.

**Working with the** **SELECT Statement**

The SELECT statement is at the heart of the SQL language. You use the SELECT statement to retrieve

data from one or more tables. Its basic syntax is

SELECT *column-list* FROM *table-list* WHERE *where-clause* ORDER BY *order-by-clause*

The SELECT clause specifies what columns you want to retrieve from the table that SQL Server

returns to the result set. The basic syntax for a SELECT clause is

SELECT *column-list*

The simplest SELECT clause looks like this:

SELECT \* FROM *Customers*

This SELECT clause, combined with the FROM clause covered next, retrieves all columns from a

table. Here's another example that retrieves only the CustomerID and CompanyName columns

from a table:

SELECT *CustomerID,* *CompanyName* FROM *Customers*

Not only can you include columns that exist in your table, but you also can include expressions in a

SELECT clause. Here's an example:

SELECT *CustomerID*, *City* + ', ' + *Region* + ' ' +

*PostalCode* AS *Address* FROM *Customers*

This SELECT clause retrieves the CustomerID column as well as an alias called Address, which

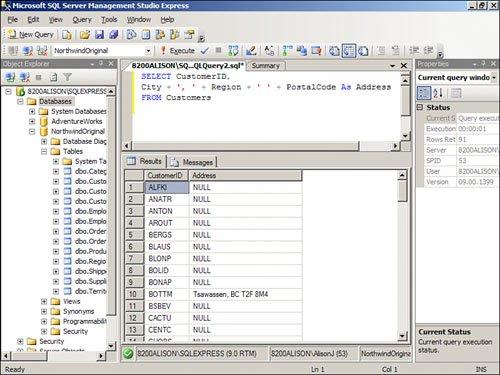
includes an expression that concatenates the City, Region, and PostalCode columns (see Figure

6.1).

**Figure 6.1. A** ***SELECT* clause that retrieves the CustomerID column as well**

**as an alias called Address, which includes an expression that**

**concatenates the City, Region, and PostalCode columns.**





**Adding on the** **FROM Clause**

The FROM clause specifies the tables or views from which the records should be selected. It can

include an alias that you use to refer to the table. The FROM clause looks like this:

FROM *table-list* [AS *alias*]

Here's an example of a basic FROM clause:

FROM *Customers*

In this case, the name of the table is Customers. If you combine the SELECT clause with the FROM

clause, the SQL statement looks like this:

SELECT *CustomerID,* *CompanyName* FROM *Customers*

This SELECT statement retrieves the CustomerID and CompanyName columns from the Customers

table.

Just as you can alias the fields included in a SELECT clause, you can also alias the tables included in

the FROM clause. The alias is used to shorten the name and to simplify a cryptic name, as well as for

a variety of other reasons. Here's an example:

SELECT *CustomerID,* *CompanyName* FROM *Customers* AS *Clients*

**Including the** **WHERE Clause**

The WHERE clause limits the records retrieved by the SELECT statement. A WHERE clause can include

columns combined by the keywords AND and O R. The syntax for a WHERE clause looks like this:

WHERE *expression1*[{AND|OR} *expression2* [...]]

A simple WHERE clause looks like this:

WHERE *Country = 'USA'*

Using an AND to further limit the criteria, the WHERE clause looks like this:

WHERE *Country* = *'USA'* AND *ContactTitle Like 'Sales%'*

This WHERE clause limits the records returned to those in which the country is equal to USA and the

ContactTitle begins with Sales. Notice that T-SQL uses the percent (%) sign as a wildcard. Using an

O R, the SELECT statement looks like this:

WHERE *Country = 'USA'* OR *Country = 'Canada'*

This WHERE clause returns all records in which the country is equal to either USA or Canada. Compare

that with the following example:

WHERE *Country = 'USA'* OR *ContactTitle* Like *'Sales%'*

This WHERE clause returns all records in which the country is equal to USA or the ContactTitle begins

with Sales. For example, the salespeople in China are returned from this WHERE clause because

their ContactTitle begins with Sales. The WHERE clause combined with the SELECT and FROM clauses

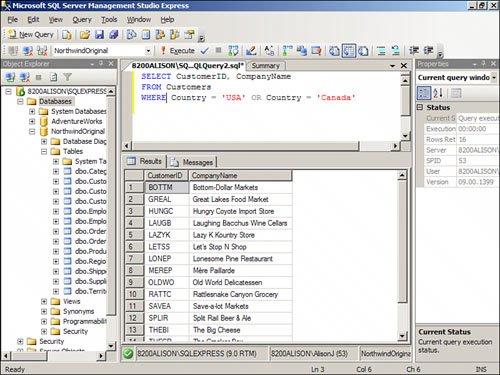
looks like this (see also Figure 6.2):

SELECT *CustomerID,* *CompanyName* FROM *Customers*

WHERE *Country = 'USA'* OR *Country = 'Canada'*

**Figure 6.2. A** ***SELECT* clause that retrieves the CustomerID and**

**CompanyName columns for all the customers in the U.S.A. and Canada.**



You must follow several rules when building a WHERE clause. You must enclose the text strings for

which you are searching in apostrophes. You must also surround dates with apostrophes. Finally,

you must include the keyword LIKE when utilizing wildcard characters. Remember that T-SQL uses

the percent symbol as the wildcard for zero or more characters. The underscore (\_) is the wildcard

for a single character.

**Using the** **ORDER BY Clause**

The ORDER BY clause determines the order in which SQL Server sorts the returned rows. It's an

optional clause and looks like this:

ORDER BY *column1*[{ASC|DESC}], *column2* [{ASC|DESC}] [,...]]

Here's an example:

ORDER BY *CustomerID*

The ORDER BY clause can include more than one field:

ORDER BY *Country,* *CustomerID*

When you specify more than one field, SQL Server uses the leftmost field as the primary level of

sort. Any additional fields are the lower sort levels. Combined with the rest of the SELECT

statement, the ORDER BY clause looks like this:

SELECT *CustomerID,* *CompanyName* FROM *Customers*

WHERE *Country = 'USA'* OR *Country = 'Canada'*

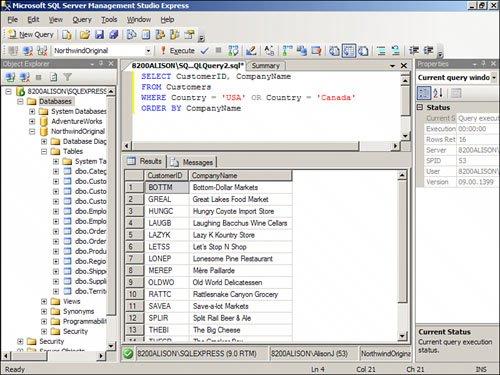
ORDER BY *CompanyName*

The results appear in order by CompanyName (see Figure 6.3).

**Figure 6.3. A** ***SELECT* clause that retrieves the CustomerID and**

**CompanyName columns for all the customers in the U.S.A. and Canada.**

**SQL Server orders the results by CompanyName.**



The ORDER BY clause enables you to determine whether the sorted output appears in ascending or

descending order. By default, output appears in ascending order. To switch to descending order,

use the optional keyword *DESC*. Here's an example:

SELECT *CustomerID,* *CompanyName* FROM *Customers* ORDER BY *CustomerID* DESC

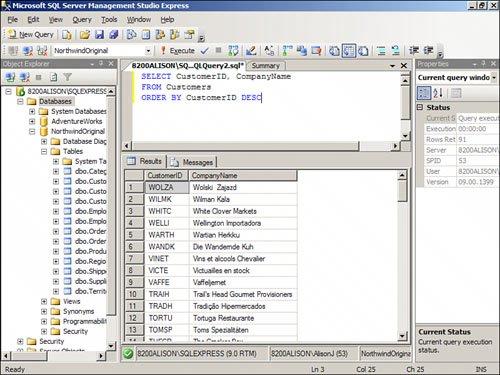
This example selects the CustomerID and CompanyName fields from the Customers table, ordering

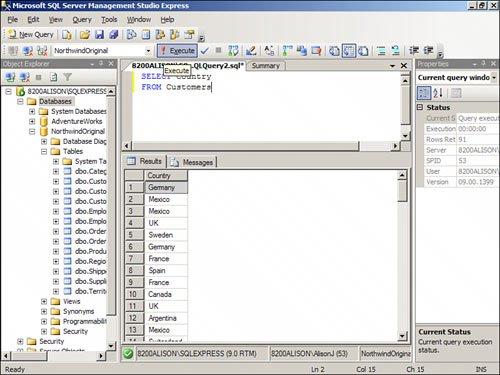
the output in descending order by the CustomerID field (see Figure 6.4).

**Figure 6.4. A** ***SELECT* clause that retrieves the CustomerID and**

**CompanyName columns for all the customers in the U.S.A. and Canada.**

**SQL Server orders the results in descending order by CustomerID.**





**Adding the** **DISTINCT Keyword**

The DISTINCT keyword ensures uniqueness of values in the column or combination of columns

included in the query result. Consider the following SQL statement:

SELECT *Country* FROM *Customers*

This statement returns one row for each customer (see Figure 6.5). The same country appears

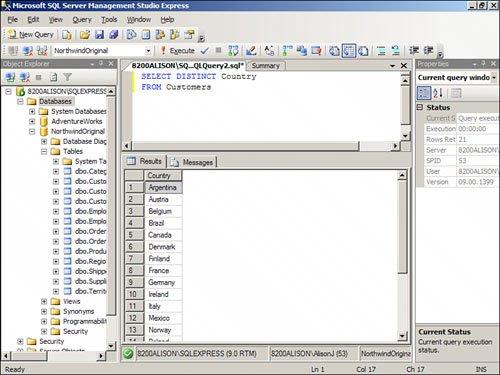
multiple times in the output.

**Figure 6.5. A** ***SELECT* statement that returns one row for each customer.**

**The same country appears multiple times in the output.**

Contrast the statement used in Figure 6.5 with this:

SELECT DISTINCT *Country* FROM *Customers*



This statement returns a list of unique countries from the list of customers (see Figure 6.6).

**Figure 6.6. A** ***SELECT* statement that returns a list of unique countries from**

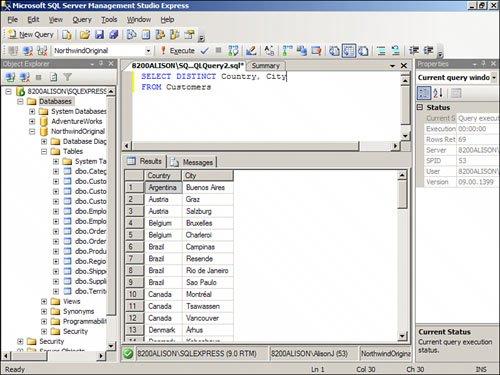
**the list of customers.**

The statement that follows returns a unique list of country and city combinations (see Figure 6.7):

SELECT DISTINCT *Country,* *City* FROM *customers*

**Figure 6.7. A** ***SELECT* statement that returns a list of unique country and**

**city combinations from the list of customers.**



**Creating Top Values Queries**

You use the TOP clause to limit the number of rows that SQL Server includes in the output. Here's

an example:

SELECT TOP 10 *OrderDate,* *Freight* FROM *Orders*

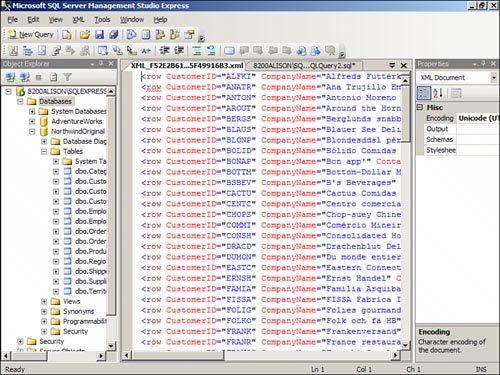
ORDER *BY Freight* DESC

This example shows the 10 highest freight amounts along with their corresponding order dates (see

Figure 6.10).

**Figure 6.10. A** ***SELECT* statement that shows the 10 order dates associated**

**with the highest 10 freight amounts.**



In addition to enabling you to select the top number of rows, T-SQL also enables you to select the

top percent of rows. Here's an example:

SELECT TOP 10 PERCENT *OrderDate,* *Freight* FROM *Orders*

ORDER BY *Freight* DESC

Here the top 10% of freight amounts appear in the query result.