SQL Server 2012 Tutorials: Writing Transact-SQL Statements

SQL Server 2012 Books Online

Quick Step-by-Step

Microsoft

SQL Server 2012 Tutorials: Writing Transact-SQL Statements

SQL Server 2012 Books Online

Summary: This tutorial is intended for users who are new to writing SQL statements. It will help new users get started by reviewing some basic statements for creating tables and inserting data. This tutorial uses Transact-SQL, the Microsoft implementation of the SQL standard. This tutorial is intended as a brief introduction to the Transact-SQL language and not as a replacement for a Transact-SQL class. The statements in this tutorial are intentionally simple, and are not meant to represent the complexity found in a typical production database.

Category: Quick Step-by-Step
Applies to: SQL Server 2012
Source: SQL Server Books Online (link to source content)
E-book publication date: June 2012

Contents

Tutorial: Writing Transact-SQL Statements

Welcome to the Writing Transact-SQL Statements tutorial. This tutorial is intended for users who are new to writing SQL statements. It will help new users get started by reviewing some basic statements for creating tables and inserting data. This tutorial uses Transact-SQL, the Microsoft implementation of the SQL standard. This tutorial is intended as a brief introduction to the Transact-SQL language and not as a replacement for a Transact-SQL class. The statements in this tutorial are intentionally simple, and are not meant to represent the complexity found in a typical production database.

Note

Novice users of databases will usually find it easier to work with SQL Server by using SQL Server Management Studio, instead of writing Transact-SQL statements.

Finding More Information

To find more information about any specific statement, either search for the statement by name in SQL Server Books Online, or use the Contents to browse the 1,800 language elements listed alphabetically under Transact-SQL Reference (Database Engine). Another good strategy for finding information is to search for key words that are related to the subject matter you are interested in. For example, if you want to know how to return a part of a date (such as the month), search the index for **dates [SQL Server]**, and then select **dateparts**. This takes you to the topic DATEPART (Transact-SQL). As another example, to find out how to work with strings, search for **string functions**. This takes you to the topic String Functions (Transact-SQL).

What You Will Learn

This tutorial shows you how to create a database, create a table in the database, insert data into the table, update the data, read the data, delete the data, and then delete the table. You will create views and stored procedures and configure a user to the database and the data.

This tutorial is divided into three lessons:

Lesson 1: Creating Database Objects

In this lesson, you create a database, create a table in the database, insert data into the table, update the data, and read the data.

Lesson 2: Configuring Permissions on Database Objects

In this lesson, you create a login and user. You will also create a view and a stored procedure, and then grant the user permission to the stored procedure.

Lesson 3: Deleting Database Objects

In this lesson, you remove access to data, delete data from a table, delete the table, and then delete the database.

Requirements

To complete this tutorial, you do not have to know the SQL language, but you should understand basic database concepts such as tables. During this tutorial, you will create a database and create a Windows user. These tasks require a high level of permissions; therefore, you should log in to the computer as an administrator.

Your system must have the following installed:

- Any edition of SQL Server.
- Either SQL Server Management Studio or Management Studio Express.
- Internet Explorer 6 or later.

Note

When you review the tutorials, we recommend that you add the **Next** and **Previous** buttons to the document viewer toolbar.

Lesson 1: Creating Database Objects

This lesson shows you how to create a database, create a table in the database, and then access and change the data in the table. Because this lesson is an introduction to using Transact-SQL, it does not use or describe the many options that are available for these statements.

Transact-SQL statements can be written and submitted to the Database Engine in the following ways:

- By using SQL Server Management Studio. This tutorial assumes that you are using Management Studio, but you can also use Management Studio Express, which is available as a free download from the Microsoft Download Center.
- By using the sqlcmd utility.
- By connecting from an application that you create.

The code executes on the Database Engine in the same way and with the same permissions, regardless of how you submit the code statements.

To run Transact-SQL statements in Management Studio, open Management Studio and connect to an instance of the SQL Server Database Engine.

This lesson contains the following topics:

Next Task in Lesson

Creating a Database (Tutorial)

Creating a Database (Tutorial)

Like many Transact-SQL statements, the CREATE DATABASE statement has a required parameter: the name of the database. CREATE DATABASE also has many optional parameters, such as the disk location where you want to put the database files. When you execute CREATE DATABASE without the optional parameters, SQL Server uses default values for many of these parameters. This tutorial uses very few of the optional syntax parameters.

Procedures

To create a database

1. In a Query Editor window, type but do not execute the following code:

   ```
   CREATE DATABASE TestData

   GO
   ```

2. Use the pointer to select the words `CREATE DATABASE`, and then press **F1**. The CREATE DATABASE topic in SQL Server Books Online should open. You can use this technique to find the complete syntax for CREATE DATABASE and for the other statements that are used in this tutorial.
3. In Query Editor, press **F5** to execute the statement and create a database named `TestData`.

When you create a database, SQL Server makes a copy of the **model** database, and renames the copy to the database name. This operation should only take several seconds, unless you specify a large initial size of the database as an optional parameter.

Note
The keyword GO separates statements when more than one statement is submitted in a single batch. GO is optional when the batch contains only one statement.

Next Task in Lesson

Creating a Table (Tutorial)

See Also

CREATE DATABASE (Transact-SQL)

Creating a Table (Tutorial)

To create a table, you must provide a name for the table, and the names and data types of each column in the table. It is also a good practice to indicate whether null values are allowed in each column.

Most tables have a primary key, made up of one or more columns of the table. A primary key is always unique. The Database Engine will enforce the restriction that any primary key value cannot be repeated in the table.

For a list of data types and links for a description of each, see Data Types (Transact-SQL).

Note
The Database Engine can be installed as case sensitive or non-case sensitive. If the Database Engine is installed as case sensitive, object names must always have the same case. For example, a table named OrderData is a different table from a table named ORDERDATA. If the Database Engine is installed as non-case sensitive, those two table names are considered to be the same table, and that name can only be used one time.

Procedures

To create a database to contain the new table

- Enter the following code into a Query Editor window.

```
USE master;
GO

--Delete the TestData database if it exists.
IF EXISTS(SELECT * from sys.databases WHERE name='TestData')
BEGIN
    DROP DATABASE TestData;
END

--Create a new database called TestData.
CREATE DATABASE TestData;
Press the F5 key to execute the code and create the database.
```

Switch the Query Editor connection to the TestData database

- In a Query Editor window, type and execute the following code to change your connection to the `TestData` database.

```
USE TestData
```

```
GO
```

To create a table

- In a Query Editor window, type and execute the following code to create a simple table named `Products`. The columns in the table are named `ProductID`, `ProductName`, `Price`, and `ProductDescription`. The `ProductID` column is the primary key of the table. `int`, `varchar(25)`, `money`, and `text` are all data types. Only the `Price` and `ProductionDescription` columns can have no data when a row is inserted or changed. This statement contains an optional element (`dbo.`) called a schema. The schema is the database object that owns the table. If you are an administrator, `dbo` is the default schema. `dbo` stands for database owner.

```
CREATE TABLE dbo.Products
    (ProductID int PRIMARY KEY NOT NULL,
     ProductName varchar(25) NOT NULL,
     Price money NULL,
     ProductDescription text NULL)
GO
```

Next Task in Lesson

Inserting and Updating Data In a Table (Tutorial)

See Also

CREATE TABLE (Transact-SQL)

Inserting and Updating Data in a Table (Tutorial)

Now that you have created the **Products** table, you are ready to insert data into the table by using the INSERT statement. After the data is inserted, you will change the content of a row by using an UPDATE statement. You will use the WHERE clause of the UPDATE statement to restrict the update to a single row. The four statements will enter the following data.

ProductID	ProductName	Price	ProductDescription
1	Clamp	12.48	Workbench clamp
50	Screwdriver	3.17	Flat head
75	Tire Bar		Tool for changing tires.
3000	3mm Bracket	.52	

The basic syntax is: INSERT, table name, column list, VALUES, and then a list of the values to be inserted. The two hyphens in front of a line indicate that the line is a comment and the text will be ignored by the compiler. In this case, the comment describes a permissible variation of the syntax.

Procedures

To insert data into a table

1. Execute the following statement to insert a row into the `Products` table that was created in the previous task. This is the basic syntax.

   ```
   -- Standard syntax
   INSERT dbo.Products (ProductID, ProductName, Price,
   ProductDescription)
       VALUES (1, 'Clamp', 12.48, 'Workbench clamp')
   GO
   ```

2. The following statement shows how you can change the order in which the parameters are provided by switching the placement of the `ProductID` and `ProductName` in both the field list (in parentheses) and in the values list.

   ```
   -- Changing the order of the columns
   INSERT dbo.Products (ProductName, ProductID, Price,
   ProductDescription)
       VALUES ('Screwdriver', 50, 3.17, 'Flat head')
   GO
   ```

3. The following statement demonstrates that the names of the columns are optional, as long as the values are listed in the correct order. This syntax is common but is not recommended because it might be harder for others to understand your code. `NULL` is specified for the `Price` column because the price for this product is not yet known.

   ```
   -- Skipping the column list, but keeping the values in order
   INSERT dbo.Products
       VALUES (75, 'Tire Bar', NULL, 'Tool for changing tires.')
   GO
   ```

4. The schema name is optional as long as you are accessing and changing a table in your default schema. Because the `ProductDescription` column allows null values and no value is being provided, the `ProductDescription` column name and value can be

dropped from the statement completely.

```
-- Dropping the optional dbo and dropping the ProductDescription
column

INSERT Products (ProductID, ProductName, Price)

    VALUES (3000, '3mm Bracket', .52)

GO
```

▶ To update the products table

1. Type and execute the following `UPDATE` statement to change the `ProductName` of the second product from `Screwdriver`, to `Flat Head Screwdriver`.

```
UPDATE dbo.Products

    SET ProductName = 'Flat Head Screwdriver'

    WHERE ProductID = 50

GO
```

Next Task in Lesson

Reading the Data in a Table (Tutorial)

See Also

INSERT (Transact-SQL)

UPDATE (Transact-SQL)

Reading the Data in a Table (Tutorial)

Use the SELECT statement to read the data in a table. The SELECT statement is one of the most important Transact-SQL statements, and there are many variations in the syntax. For this tutorial, you will work with five simple versions.

Procedures

▶ To read the data in a table

1. Type and execute the following statements to read the data in the `Products` table.

```
-- The basic syntax for reading data from a single table

SELECT ProductID, ProductName, Price, ProductDescription

    FROM dbo.Products

GO
```

2. You can use an asterisk to select all the columns in the table. This is often used in ad

hoc queries. You should provide the column list in you permanent code so that the statement will return the predicted columns, even if a new column is added to the table later.

```
-- Returns all columns in the table
-- Does not use the optional schema, dbo
SELECT * FROM Products
GO
```

3. You can omit columns that you do not want to return. The columns will be returned in the order that they are listed.

```
-- Returns only two of the columns from the table
SELECT ProductName, Price
    FROM dbo.Products
GO
```

4. Use a `WHERE` clause to limit the rows that are returned to the user.

```
-- Returns only two of the records in the table
SELECT ProductID, ProductName, Price, ProductDescription
    FROM dbo.Products
    WHERE ProductID < 60
GO
```

5. You can work with the values in the columns as they are returned. The following example performs a mathematical operation on the `Price` column. Columns that have been changed in this way will not have a name unless you provide one by using the `AS` keyword.

```
-- Returns ProductName and the Price including a 7% tax
-- Provides the name CustomerPays for the calculated column
SELECT ProductName, Price * 1.07 AS CustomerPays
    FROM dbo.Products
GO
```

Functions That Are Useful in a SELECT Statement

For information about some functions that you can use to work with data in SELECT statements, see the following topics:

String Functions (Transact-SQL)	Date and Time Functions (Transact-SQL)
Mathematical Functions (Transact-SQL)	Text and Image Functions (Transact-SQL)

Next Task in Lesson

Summary: Creating Database Objects

See Also

SELECT (Transact-SQL)

Summary: Creating Database Objects

In this tutorial you have created a database, created a table in the database, inserted data into the table, changed the data, and then read the data from the table. The syntax for the statements that were used is only the basic syntax and many syntax options were not covered in this tutorial. To learn more about these statements, read the complete syntax for the statements in SQL Server Books Online, and review the many examples that are provided in those topics.

Next Lesson

Lesson 2: Configuring Permissions on Database Objects

See Also

CREATE DATABASE (Transact-SQL)

Lesson 2: Configuring Permissions on Database Objects

Granting a user access to a database involves three steps. First, you create a login. The login lets the user connect to the SQL Server Database Engine. Then you configure the login as a user in the specified database. And finally, you grant that user permission to database objects. This lesson shows you these three steps, and shows you how to create a view and a stored procedure as the object.

This lesson contains the following topics:

Next Task in Lesson

Creating a login

Creating a Login

To access the Database Engine, users require a login. The login can represent the user's identity as a Windows account or as a member of a Windows group, or the login can be a SQL Server login that exists only in SQL Server. Whenever possible you should use Windows Authentication.

By default, administrators on your computer have full access to SQL Server. For this lesson, we want to have a less privileged user; therefore, you will create a new local Windows Authentication account on your computer. To do this, you must be an administrator on your computer. Then you will grant that new user access to SQL Server.

Procedures

To create a new Windows account

1. Click **Start**, click **Run**, in the **Open** box, type **%SystemRoot%\system32\compmgmt.msc /s**, and then click **OK** to open the Computer Management program.
2. Under **System Tools**, expand **Local Users and Groups**, right-click **Users**, and then click **New User**.
3. In the **User name** box type **Mary**.
4. In the **Password** and **Confirm password** box, type a strong password, and then click **Create** to create a new local Windows user.

To create a login

1. In a Query Editor window of SQL Server Management Studio, type and execute the following code replacing `computer_name` with the name of your computer. `FROM WINDOWS` indicates that Windows will authenticate the user. The optional `DEFAULT_DATABASE` argument connects `Mary` to the `TestData` database, unless her connection string indicates another database. This statement introduces the semicolon as an optional termination for a Transact-SQL statement.

   ```
   CREATE LOGIN [computer_name\Mary]
       FROM WINDOWS
       WITH DEFAULT_DATABASE = [TestData];
   GO
   ```

   This authorizes a user name `Mary`, authenticated by your computer, to access this instance of SQL Server. If there is more than one instance of SQL Server on the computer, you must create the login on each instance that `Mary` must access.

Note

Because `Mary` is not a domain account, this user name can only be authenticated on this computer.

Next Task in Lesson

Granting Access to a Database

See Also

CREATE LOGIN (Transact-SQL)

Authentication Mode

Granting Access to a Database

Mary now has access to this instance of SQL Server, but does not have permission to access the databases. She does not even have access to her default database **TestData** until you authorize her as a database user.

To grant Mary access, switch to the **TestData** database, and then use the CREATE USER statement to map her login to a user named Mary.

Procedures

To create a user in a database

1. Type and execute the following statements (replacing `computer_name` with the name of your computer) to grant `Mary` access to the `TestData` database.

```
USE [TestData];
GO

CREATE USER [Mary] FOR LOGIN [computer_name\Mary];
GO
```

Now, Mary has access to both SQL Server and the `TestData` database.

Next Task in Lesson

Creating a Stored Procedure

Creating Views and Stored Procedures

Now that Mary can access the **TestData** database, you may want to create some database objects, such as a view and a stored procedure, and then grant Mary access to them. A view is a stored SELECT statement, and a stored procedure is one or more Transact-SQL statements that execute as a batch.

Views are queried like tables and do not accept parameters. Stored procedures are more complex than views. Stored procedures can have both input and output parameters and can contain statements to control the flow of the code, such as IF and WHILE statements. It is good programming practice to use stored procedures for all repetitive actions in the database.

For this example, you will use CREATE VIEW to create a view that selects only two of the columns in the **Products** table. Then, you will use CREATE PROCEDURE to create a stored procedure that accepts a price parameter and returns only those products that cost less than the specified parameter value.

Procedures

To create a view

1. Execute the following statement to create a very simple view that executes a select statement, and returns the names and prices of our products to the user.

```
CREATE VIEW vw_Names

   AS

   SELECT ProductName, Price FROM Products;

GO
```

Test the view

1. Views are treated just like tables. Use a `SELECT` statement to access a view.

```
SELECT * FROM vw_Names;

GO
```

To create a stored procedure

1. The following statement creates a stored procedure name `pr_Names`, accepts an input parameter named `@VarPrice` of data type `money`. The stored procedure prints the statement `Products less than` concatenated with the input parameter that is changed from the `money` data type into a `varchar(10)` character data type. Then, the procedure executes a `SELECT` statement on the view, passing the input parameter as part of the `WHERE` clause. This returns all products that cost less than the input

parameter value.

```
CREATE PROCEDURE pr_Names @VarPrice money
   AS
   BEGIN
      -- The print statement returns text to the user
      PRINT 'Products less than ' + CAST(@VarPrice AS
varchar(10));
      -- A second statement starts here
      SELECT ProductName, Price FROM vw_Names
            WHERE Price < @varPrice;
   END
GO
```

Test the stored procedure

1. To test the stored procedure, type and execute the following statement. The procedure should return the names of the two products entered into the `Products` table in Lesson 1 with a price that is less than `10.00`.

```
EXECUTE pr_Names 10.00;
GO
```

Next Task in Lesson

Granting Access to a Database Object

See Also

CREATE VIEW (Transact-SQL)

CREATE PROCEDURE (Transact-SQL)

Granting Access to a Database Object

As an administrator, you can execute the SELECT from the **Products** table and the **vw_Names** view, and execute the **pr_Names** procedure; however, Mary cannot. To grant Mary the necessary permissions, use the GRANT statement.

Procedures

Procedure Title

1. Execute the following statement to give `Mary` the `EXECUTE` permission for the `pr_Names` stored procedure.

   ```
   GRANT EXECUTE ON pr_Names TO Mary;
   GO
   ```

In this scenario, Mary can only access the **Products** table by using the stored procedure. If you want Mary to be able to execute a SELECT statement against the view, then you must also execute `GRANT SELECT ON vw_Names TO Mary`. To remove access to database objects, use the REVOKE statement.

Note
If the table, the view, and the stored procedure are not owned by the same schema, granting permissions becomes more complex.

About GRANT

You must have EXECUTE permission to execute a stored procedure. You must have SELECT, INSERT, UPDATE, and DELETE permissions to access and change data. The GRANT statement is also used for other permissions, such as permission to create tables.

Next Task in Lesson

Summary: Configuring Permissions on Database Objects

See Also

GRANT (Transact-SQL)

REVOKE (Transact-SQL)

Summary: Configuring Permissions on Database Objects

Logins give users permissions to connect to SQL Server. Users are logins that can access a specific database. Use the GRANT statement to give users permission to read and to access and change the data.

A view is a single SELECT statement and looks like a table to the user. A stored procedure is one or more Transact-SQL statements that execute as a batch.

Next Lesson in Tutorial

Lesson 3: Deleting Database Objects

Lesson 3: Deleting Database Objects

This short lesson removes the objects that you created in Lesson 1 and Lesson 2, and then drops the database.

This lesson contains one topic:

- Deleting Database Objects

Next Task in Lesson

Deleting Database Objects

Deleting Database Objects

To remove all traces of this tutorial, you could just delete the database. However, in this topic, you will go through the steps to reverse every action you took doing the tutorial.

Procedures

Removing permissions and objects

1. Before you delete objects, make sure you are in the correct database:

   ```
   USE TestData;
   GO
   ```

2. Use the `REVOKE` statement to remove execute permission for `Mary` on the stored procedure:

   ```
   REVOKE EXECUTE ON pr_Names FROM Mary;
   GO
   ```

3. Use the `DROP` statement to remove permission for `Mary` to access the `TestData` database:

   ```
   DROP USER Mary;
   GO
   ```

4. Use the `DROP` statement to remove permission for `Mary` to access this instance of SQL Server 2005:

   ```
   DROP LOGIN [<computer_name>\Mary];
   GO
   ```

5. Use the `DROP` statement to remove the store procedure `pr_Names`:

```
DROP PROC pr_Names;
GO
```

6. Use the `DROP` statement to remove the view `vw_Names`:

   ```
   DROP View vw_Names;
   GO
   ```

7. Use the `DELETE` statement to remove all rows from the `Products` table:

   ```
   DELETE FROM Products;
   GO
   ```

8. Use the `DROP` statement to remove the `Products` table:

   ```
   DROP Table Products;
   GO
   ```

9. You cannot remove the `TestData` database while you are in the database; therefore, first switch context to another database, and then use the `DROP` statement to remove the `TestData` database:

   ```
   USE MASTER;
   GO
   DROP DATABASE TestData;
   GO
   ```

This concludes the Writing Transact-SQL Statements tutorial. Remember, this tutorial is a brief overview and it does not describe all the options to the statements that are used. Designing and creating an efficient database structure and configuring secure access to the data requires a more complex database than that shown in this tutorial.

Return to SQL Server Tools Portal

Tutorial: Writing Transact-SQL Statements

See Also

REVOKE (Transact-SQL)

DROP USER (Transact-SQL)

DROP LOGIN (Transact-SQL)

DROP PROCEDURE (Transact-SQL)

DROP VIEW (Transact-SQL)
DELETE (Transact-SQL)
DROP TABLE (Transact-SQL)
DROP DATABASE (Transact-SQL)