





Lesson 1: Create and query database objects

 docs.microsoft.com/en-us/sql/t-sql/lesson-1-creating-database-objects

- 07/29/2018
- 10 minutes to read
- Contributors

APPLIES TO:  SQL Server (starting with 2008)  Azure SQL Database  Azure SQL Data Warehouse  Parallel Data Warehouse

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.

Prerequisites

To complete this tutorial, you need SQL Server Management Studio and access to a SQL Server instance.

Install [SQL Server Management Studio](#).

If you don't have access to a SQL Server instance, select your platform from the following links. If you choose SQL Authentication, use your SQL Server login credentials.

- **Windows:** [Download SQL Server 2017 Developer Edition](#).
- **macOS:** [Download SQL Server 2017 on Docker](#).

Create a database

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.

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

SQL

```
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.

Create a Table

APPLIES TO:  SQL Server (starting with 2008)  Azure SQL Database  Azure SQL Data Warehouse  Parallel Data Warehouse

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. To create a table, you must have the `CREATE TABLE` permission, and the `ALTER SCHEMA` permission on the schema that will contain the table. The `db_ddladmin` fixed database role has these permissions.

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.

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.

SQL

```
USE TestData
GO
```

Create the 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 **ProductDescription** 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.

SQL

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

Insert and update data in a table

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

ProductID	ProductName	Price	ProductDescription
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.

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.

SQL

```
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.

SQL

```
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.

SQL

```
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.

SQL

```
INSERT Products (ProductID, ProductName, Price)
VALUES (3000, '3mm Bracket', .52)
GO
```

Update the products table

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

SQL

```
UPDATE dbo.Products
SET ProductName = 'Flat Head Screwdriver'
WHERE ProductID = 50
GO
```

Read data from a table

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.

Read the data in a table

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

SQL

```
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.

SQL

```
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.

SQL

```
SELECT ProductName, Price
FROM dbo.Products
GO
```

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

SQL

```
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.

SQL

```
SELECT ProductName, Price * 1.07 AS CustomerPays
FROM dbo.Products
GO
```

Useful functions in a SELECT Statement

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

Create views and stored procedures

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.

Create a view

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.

SQL

```
CREATE VIEW vw_Names
AS
SELECT ProductName, Price FROM Products;
GO
```

Test the view

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

SQL

```
SELECT * FROM vw_Names;
GO
```

Create a stored procedure

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.

SQL

```
CREATE PROCEDURE pr_Names @VarPrice money
AS
BEGIN

    PRINT 'Products less than ' + CAST(@VarPrice AS varchar(10));

    SELECT ProductName, Price FROM vw_Names
        WHERE Price < @varPrice;

END
GO
```

Test the stored procedure

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` .

SQL

```
EXECUTE pr_Names 10.00;  
GO
```

Next steps

The next article teaches you how to configure permissions on database objects. The objects created in lesson 1 will also be used in lesson 2.

Go to the next article to learn more:

[Next steps](#)