**Lab 1: SQL Introduction / Recap**

To complete this tutorial, you need SQL Server Management Studio and access to a SQL Server instance. [This exercise based on Microsoft T-SQL syntax and MS SQL Server Environment

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.

**NOTE : All your tasks are highlighted in Yellow**

Create a database

1. In a Query Editor window, type but don't execute the following code:

CREATE DATABASE TestData

GO

1. Use the pointer to select the words CREATE DATABASE, and then press **F1**. The CREATE DATABASE article 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.
2. 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

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 enforces the restriction that any primary key value can't be repeated in the table.

For a list of data types and links for a description of each, see [Data Types (Transact-SQL)](https://learn.microsoft.com/en-us/sql/t-sql/data-types/data-types-transact-sql?view=sql-server-ver16).

**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.

USE TestData

GO

### Create the table

In a Query Editor window, type and execute the following code to create a 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 varchar(max) 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 varchar(max) 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 use the WHERE clause of the UPDATE statement to restrict the update to a single row. The four statements 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 | 3 mm Bracket | 0.52 |  |

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 is 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.

-- Standard syntax

INSERT dbo.Products (ProductID, ProductName, Price, ProductDescription)

VALUES (1, 'Clamp', 12.48, 'Workbench clamp')

GO

If the insert succeeds, proceed to the next step.

If the insert fails, it may be because the Product table already has a row with that product ID in it. To proceed, delete all the rows in the table and repeat the preceding step. [TRUNCATE TABLE](https://learn.microsoft.com/en-us/sql/t-sql/statements/truncate-table-transact-sql?view=sql-server-ver16) deletes all the rows in the table.

Run the following command to delete all the rows in the table:

TRUNCATE TABLE TestData.dbo.Products;

GO

After you truncate the table, repeat the INSERT command in this step.

1. 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

1. 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 isn't 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 isn't 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

1. 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, '3 mm Bracket', 0.52)

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.

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 basic versions.

### 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

1. You can use an asterisk (\*) to select all the columns in the table. The asterisk is for ad hoc queries. In permanent code, provide the column list so that the statement returns 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

1. You can omit columns that you don't want to return. The columns are returned in the order that they are listed.

-- Returns only two of the columns from the table

SELECT ProductName, Price

FROM dbo.Products

GO

1. 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

1. 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 don't 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

### 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 articles:

[String Functions (Transact-SQL)](https://learn.microsoft.com/en-us/sql/t-sql/functions/string-functions-transact-sql?view=sql-server-ver16)

[Date and Time Data Types and Functions (Transact-SQL)](https://learn.microsoft.com/en-us/sql/t-sql/functions/date-and-time-data-types-and-functions-transact-sql?view=sql-server-ver16)

[Mathematical Functions (Transact-SQL)](https://learn.microsoft.com/en-us/sql/t-sql/functions/mathematical-functions-transact-sql?view=sql-server-ver16)

[Text and Image Functions (Transact-SQL)](https://learn.microsoft.com/en-us/sql/t-sql/functions/text-and-image-functions-textptr-transact-sql?view=sql-server-ver16)

## 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 don't 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 use CREATE VIEW to create a view that selects only two of the columns in the Products table. Then, you 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 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

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

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.

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

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