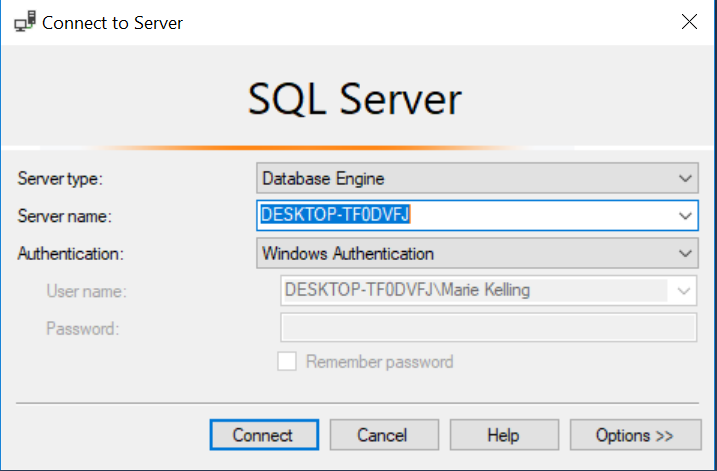
SQL Server Tutorial

SQL Server Management Studio and SQL Server Database are 2 different things

* SQL Server Management Studio is a just a client tool to connect to the SQL Database
* must configure / specify how SSMS should connect to the SQL Database
* ‘.’ Or ‘(local)’ – specifies that you want to connect to the local server



**PART 2 – Creating and Altering a Database**:

SQL Server DB can be created, altered, and dropped

* Using SSMS
* Using a Query
* CREATE DATABASE \_\_\_\_
* ALTER DATABASE \_\_\_\_ MODIFY NAME = \_\_\_\_\_
* DROP DATABASE \_\_\_\_

\*Cannot drop DB if currently in use –

* Must change from Multi-user mode to Single-user Mode
* ALTER DATABASE \_\_\_\_ SET SINGLE\_USER WITH ROLLBACK IMMEDIATE;

RC DB 🡪 Properties 🡪 Files – path contains 2 files associated w/ the newly created DB – MDF & LDF

**PART 3 – Creating and working with Tables**:

Foreign Keys:

* SSMS
* RC Table 🡪 Design – RC FK 🡪 Relationships 🡪 Add 🡪 T & C Specification
* Query
* Alter Table \_\_\_\_ Add Constraint tblPerson\_GenderID\_FK

FOREIGN KEY (GenderId) references tblGender (ID)

**PART 4 – Adding a Default Constraint**:

**PART 5 – Cascading Referential Integrity**:

Options for setting up Cascading Referential Integrity Constraint – Insert & Update Specifications

* No Action
* Cascade
* Set Null
* Set Default

**PART 6 – Adding a Check Constraint**:

* Used to limit the range of the values that can be entered for a column

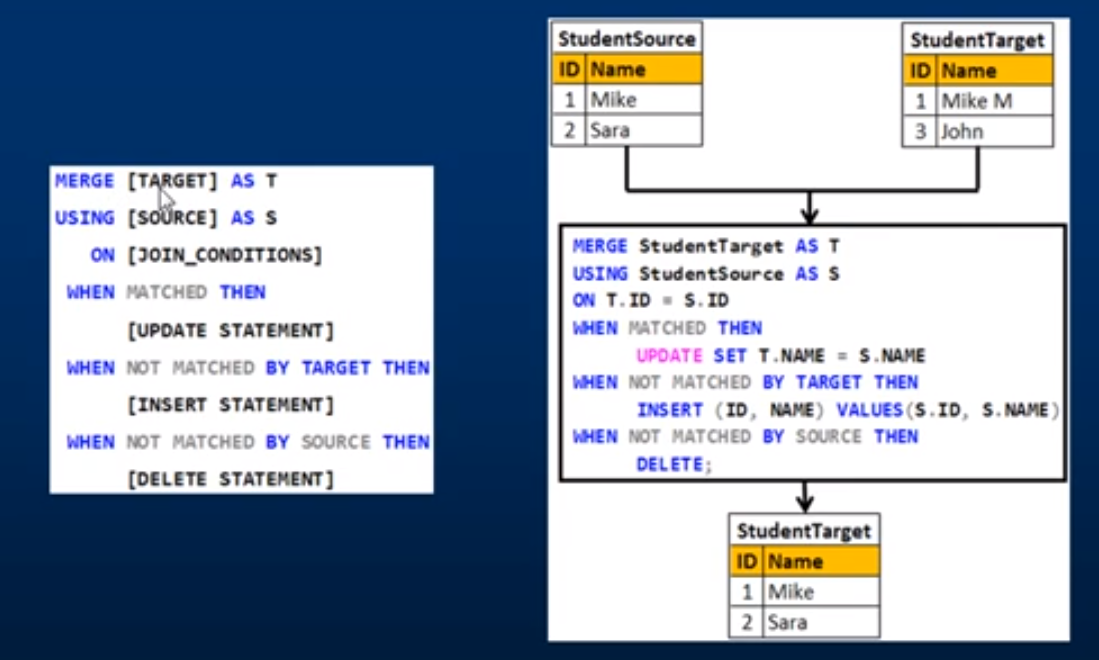
**PART 7 – Identity Columns**:

Marking a column as an identity column means you want SQL SERVER to automatically generate it

**PART 18 – Stored Procedures**:

**Merge Statement**:

* \*Combines Insert, Update, Delete into one statement
* Records that are matched – each table has a record with the same ID (primary key)
* Records that aren’t matched – record present in the source table that is not in the target table



Inserting new records – need to implement auto incrementation:

* Open table in Design 🡪 select column 🡪 Column Properties 🡪 Identity Specification = yes
* ‘saving changes not permitted’ 🡪 in SSMS 🡪 Tools 🡪 Options 🡪 Designers
* Uncheck the option “Prevent saving changes that require table re-creation”
* This may result in bugs or lost information – safer route may be to:
* \*\*use ALTER TABLE

**Parameters**:

The @CustID means it's a parameter that you will supply a value for later in your code. This is the best way of protecting against SQL injection. Create your query using parameters, rather than concatenating strings and variables. The database engine puts the parameter value into where the placeholder is, and there is zero chance for SQL injection.

6-29-18:

Linking Access DB table to SQL DB table?

* In Access – File 🡪 Get External Data 🡪 Link Tables
* Browse ‘File of Type’ 🡪 select ODBC Databases()
* ….

\*how to continually update a sql table based off another table\* google this

Importing data from Excel sheet to SQL table

Making a table column a ‘Computed Column Specification’

If you don't already know how to install a custom VBA  
macro or function in your workbook, here's how:  
  
1) Go to the Visual Basic Editor (VBE) using the menu  
Tools -> Macros -> Visual Basic Editor.  
  
2) In the VBE, use the Insert -> Module menu to add  
a module to your workbook's VBA project.  
  
3) An empty code window pane will appear in the upper  
right portion of the VBE. Paste the code you want  
to install here. You can insert multiple Sub and/  
or Function macros in this pane in the same module.  
  
4) These macros will immediately become available for  
use in the Workbook within which they have been   
copied. To run the macro, go to Tools > Macro >   
Macros... > select the macro and Run. You can also  
assign the macro to a commandbutton or custom  
toolbar button.

**SQL Basics**:

* Statement:
* Any SQL command such as SELECT, INSERT, UPDATE, DELETE
* May have a persistent effect on schemas or data, or which may control, transactions, program flow, connections, sessions, or diagnostics
* Query:
* Retrieve Data based on specific criteria
* Synonym for SELECT Statement
* **View** is a virtual table, a snap shot of the database, which is based on SQL SELECT query
* It references one or more existing database tables or other views
* **Stored Procedure** is a group of Transact-SQL statements compiled into a single execution plan
* **SQL** - Using **Views**. A **view** is nothing more than a **SQL** statement that is stored in the database with an associated name. ... A **view can** contain all rows of a table or **select** rows from a table. A **view can** be created from **one** or many tables which depends on the written **SQL query** to create a **view**.
* A **view** is a virtual **table**. A **view** consists of rows and columns just like a **table**. The **difference between** a **view** and a **table** is that views are definitions built on top of other tables (or views), and do not hold data themselves. If data is changing **in the** underlying **table**, the same change is reflected **in the view**.
* Using a view saves you copying and pasting your queries and adds code reusability, so you can change a single view instead of 10 queries in the different places of your code.
* Different permissions can be granted on views and tables, so that you can show only a portion of data to a user
* A view can be materialized, which means caching the results of the underlying query

SQL Server is very efficient in operating on data in sets

* Update, Delete, Select, etc. all operate on records in sets (based off condition(s))
* When there is a need to process the rows on a row-by-row basis use CURSORS

Cursor:

* A pointer to a row
* Declare a cursor for a result set: DECLARE cursor CURSOR FOR select statement
* The cursor will be pointing to that result set returned by the select statement
* OPEN cursor
* Will execute the select statement associated with the cursor
* FETCH NEXT FROM cursor INTO @param, @param
* Retrieves the values from the set and inserts them into the given SQL parameters
* WHILE (@@FETCH\_STATUS = 0)
* Is 0 as long as there are more rows to retrieve
* Once gone through all rows in set – the status will no longer be 0

BEGIN

PRINT ‘id = ’ + CAST(@ProductId as Nvarchar(10)) + ‘ Name = ’ + @Name

FETCH NEXT FROM cursor INTO @param, @param

END

* CLOSE cursor
* DEALLOCATE cursor