



This SQL Workshop looks to how we can develop a SQL Server Database with multiple related Tables and examines how we can Insert data, generate Views and Query data across related tables. We will start with a single table and demonstrate how we scale the solution into a multi related tables that meet the needs of a mobile app or website. How we can generate Views across related data. This series covers how we can generate Views across by utilising Joins for related Tables. Subqueries are discussed as well as group by constructs.

|  |  |
| --- | --- |
|  | MS SQL Server – SQL Server Guide 04: |
|  |  |
|  |  |
|  |  |
|  |  | Author: Mansha Nawaz |
|  | Using SQL Server Management Studio SSMS  Author: Mansha Nawaz |

How to Build DB Queries and Views - Music Example

Microsoft SQL Server is a Relational Database Management System (RDBMS)

developed by Microsoft. It is a highly scalable product that can be run on anything from

a single laptop, to a network of high-powered cloud servers, and anything in between.

This **SQL Quick Guide** demonstrates how wecan **create** a **database** with 3 related **tables, insert data, generate views** and perform some basic **queries** across related tables in **SQL Server Management Studio** ([**SSMS**](https://docs.microsoft.com/en-us/sql/ssms/quickstarts/ssms-connect-query-sql-server?view=sql-server-ver15)) GUI.

The SQL Server skills to acquire from the SQL Quick Guides are:

SQL Quick Guide 01 covers the following:

* Installing SQL Server and SSMS for home use
* How to login to SQL Server Management Studio SSMS
* How to Navigate SQL Server Management Studio SSMS
* Create a Database and a Table
* Create a Table and replicate or amend the TSQL DDL Code
* Add Constraints if required.
* Insert data into the table.
* Query the data by selecting records.

SQL Quick Guide 02: covers the following:

* Create a Tables via SSMS GUI
* Create a Tables using SQL Script Code
* Create a Relationship between many Tables.
* Insert Data and Import Data

**SQL Quick Guide 03: covers the following:**

* **Query Designer**
* **Create a View and the View Designer**
* **Save Query Results to CSV File**
* ***Script a Database – separate guide!***

Content Overview:

[How to Build DB Queries and Views - Music Example- Music Example 0](file:///D:\lectures\_2020%20Covid%20Lectures\2021%20SQL%20Server\MNSQL-QuickGuide04%20-%20How%20to%20Build%20DB%20Queries%20and%20Views%20-%20Music%20Example.docx#_Toc77786023)

[1. Use the Query Designer to build complex queries across multiple tables without writing any code. 2](#_Toc77786024)

[1.1. Use the Query Designer to Build a Simple Query 2](#_Toc77786025)

[1.2. Open the Query Designer 3](#_Toc77786026)

[1.3. Add the Tables. 4](#_Toc77786027)

[1.4. Design the Query 5](#_Toc77786028)

[1.5. How to Design a Query 5](#_Toc77786029)

[1.6. Our Example 6](#_Toc77786030)

[2. Create a View from multiple tables 7](#_Toc77786031)

[2.1. Benefits of Views 7](#_Toc77786032)

[2.2. How to Create a View 7](#_Toc77786033)

[2.3. Design the View 8](#_Toc77786034)

[2.4. Sample Code 8](#_Toc77786035)

[2.5. Execute the View. 9](#_Toc77786036)

[2.6. Query the View 10](#_Toc77786037)

[2.7. Filter the View 11](#_Toc77786038)

[2.8. Alter a View 11](#_Toc77786039)

[2.8.1. Design the Altered View 11](#_Toc77786040)

[2.8.2. Sample Code 12](#_Toc77786041)

[2.8.3. Query the View 13](#_Toc77786042)

[3. Generating SQL Create / Insert Scripts Revisited 14](#_Toc77786043)

[3.1. Generating Database Script 14](#_Toc77786044)

[3.1. Recreating the database at home – method 1 18](#_Toc77786045)

[3.2. Recreating the database at home – method 2 18](#_Toc77786046)

[3.3. Recreating the database at Uni 19](#_Toc77786047)

[3.4. Alternative method of loading and running the SQL generated Creates and Schemas 19](#_Toc77786048)

# **Use the Query Designer to build complex queries across multiple tables without writing any co**de.

SQL Server Management Studio includes the Query Designer to assist in building queries. It is a visual tool that allows you to select the tables and columns you want in your query, as well as any filtering criteria.

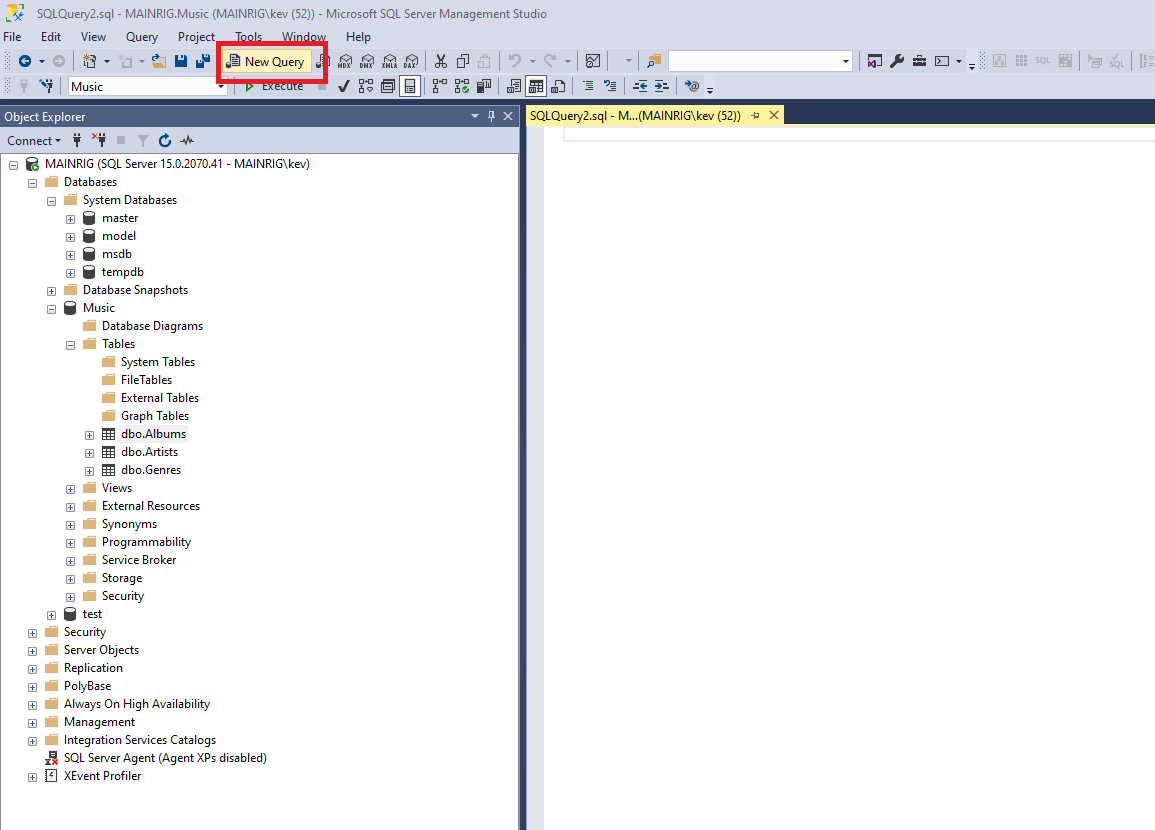
No need to write any SQL code — the Query Designer will generate that for you.

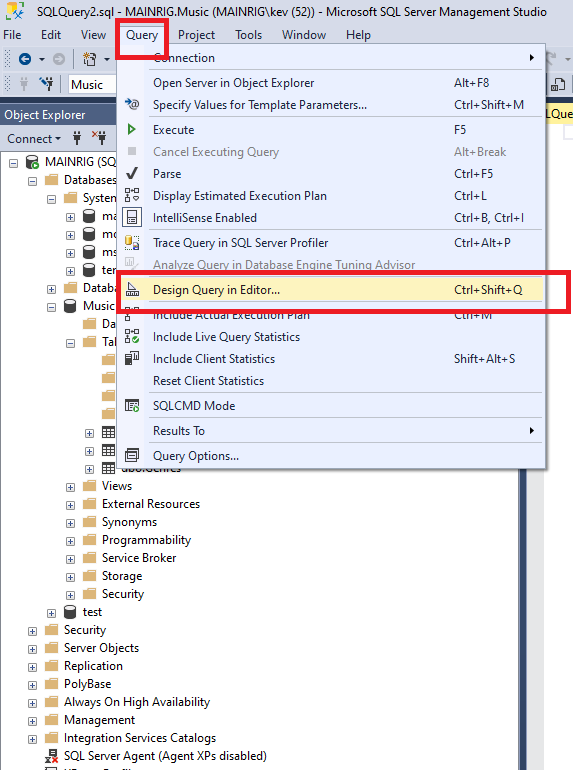
# Use the Query Designer to Build a Simple Query

We will now use the Query Designer to build a simple query. If you have been following Quick Guides you will now have a database with three tables — all of which contain data.

Since we have also established a relationship between these tables, we can now run queries across all three, returning related records.

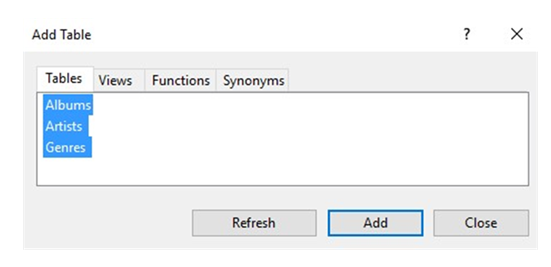
# Open the Query Designer





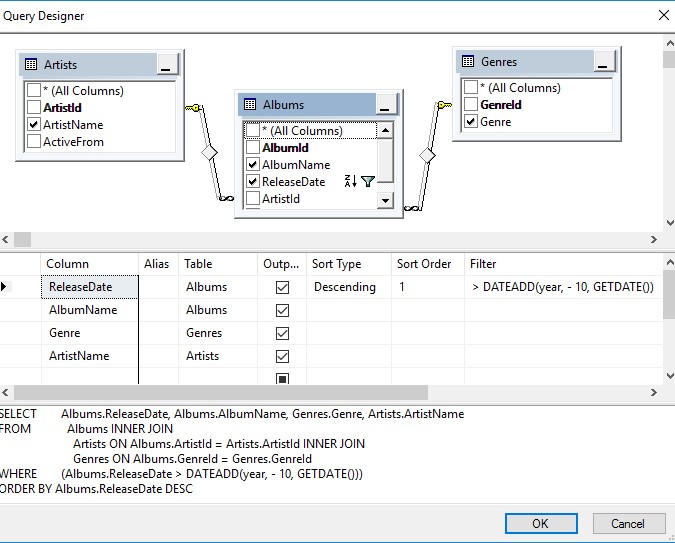
* Open a new query window by clicking on **New Query** in the toolbar.
* Then select Query > Design Query in Editor... from the top menu.
* If you can't see the Query option in the top menu, click inside the query window first. This will change the top menu items to be query-related options.

# Add the Tables.



* Here, you select which tables you would like in your query.
* Select all three and click **Add.**
* Click **Close** to close the dialog box.

# Design the Query



You will now see the selected tables, and their relationships, in the Query Designer. Feel free to click and drag them around to provide a better visualization of their relationship with each other. You can also re-size each pane by clicking its edge and dragging it up or down.

# How to Design a Query

In the top pane (the Diagram Pane), click each column that you want to include in the query (whether you want to display it or not). Each column you select in the top pane will automatically appear in the middle pane.

* In the middle pane (the Grid Pane or Criteria Pane), use the **Output** checkbox to indicate which columns will be returned in the results.
* Use **Sort Type** to specify the order of the results by a given column. You can use Sort Order to specify which column will be sorted first, second, etc.
* Use Filter to add filtering criteria to **filter** the records returned.

The bottom pane (the SQL Pane) dynamically generates the SQL statement that your query produces. This is the statement that will be run when you close the Query Designer and execute the query.

# Our Example

In our example, our query will return all albums (along with their genre, artist, and genre) that were released in the last ten years.

* The criteria to achieve this is **>DATEADD(year, - 10, GETDATE()**).
* The query will sort the results by the release date in descending order.

Here's a close-up of the Criteria Pane:



* You can change the order of the columns by clicking and dragging them up or down.
* If you're reading this long after this tutorial was written, you might need to adjust the criteria to go back 20 years or more before you get any results.
* Alternatively, you could add something a bit more modern to the music collection :)

Make sure you keep this query open in the query window because next, we will save it as a view in the next Guide.

The Query Code generated has also been provided below:

|  |
| --- |
| SELECT Albums.ReleaseDate,  Albums.AlbumName,  Genres.Genre, Artists.ArtistName  FROM Albums  INNER JOIN  Artists ON  Albums.ArtistId = Artists.ArtistId  INNER JOIN  Genres ON  Albums.GenreId = Genres.GenreId  WHERE (Albums.ReleaseDate > DATEADD(year, - 10, GETDATE())); |

# Create a View from multiple tables

In SQL Server, you can query as **views**. **Views** are beneficial for many reasons, including security, usability, and convenience.

In SQL Server, a ***view*** is a virtual table whose contents are defined by a query. It is basically a pre-written query that is stored on the database.

A view consists of a **Select**statement, and when you run a query against the view, you see the results of it like you would when opening a table. Views are referred to as virtual tables because they can pull together data from multiple tables, as well as aggregate data, and present it as though it is a single table.

# Benefits of Views

A view can be useful when there are multiple users with different levels of access, who all need to see portions of the data in the database (but not necessarily all the data). Views can do the following:

* Restrict access to specific rows in a table.
* Restrict access to specific columns in a table.
* Join columns from multiple tables and present them as though they are part of a single table.
* Present aggregate information (such as the results of the COUNT() function)

# How to Create a View

You create a view by using the CREATE VIEW statement, followed by the SELECT statement.

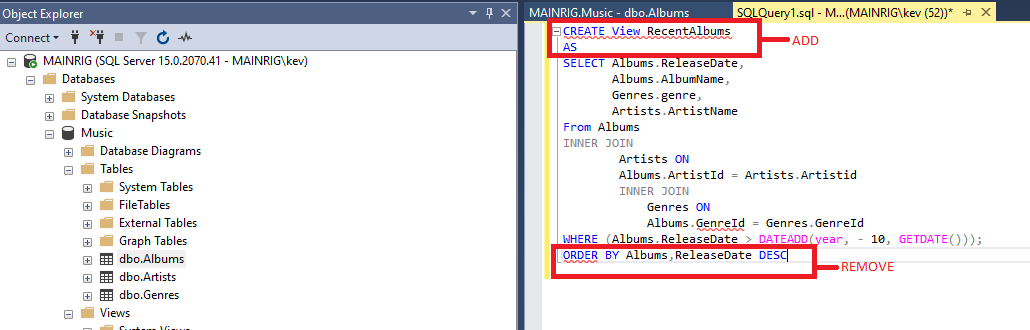
CREATE VIEW ViewName

AS

SELECT ...

We'll now create a view from our previous **query** from the previous section[.](http://www.quackit.com/sql_server/sql_server_2016/tutorial/query_designer_in_sql_server_2016.cfm)

# Design the View



* Take the query from our previous example, and prefix it with CREATE VIEW RecentAlbums AS
* Also remove the **Orderby** clause, because **views** do not support this clause (unless TOP, OFFSET or FOR XML is also specified).
* Also add a semicolon to the end of the statement, as a statement terminator (more on that below).

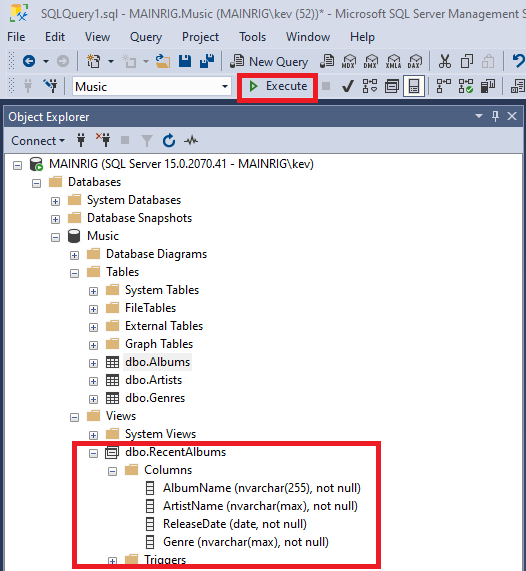
# Sample Code

Below is the code from our example, with the ORDER BY clause removed, and a semicolon added as a statement terminator. I have also reformatted it a bit to make it more readable).

|  |
| --- |
| -- Task: (1) **Click** the **New Query** tab:  -- (2) **Copy** n **Paste** the below **TSQL** code into a new **Query Window.**  -- (3) Click on E**xecute** to run the **TSQL**: |
| **CREATE VIEW RecentAlbums**  **AS**  SELECT Albums.ReleaseDate,  Albums.AlbumName,  Genres.Genre, Artists.ArtistName  FROM Albums  INNER JOIN  Artists ON  Albums.ArtistId = Artists.ArtistId  INNER JOIN  Genres ON  Albums.GenreId = Genres.GenreId  WHERE (Albums.ReleaseDate > DATEADD(year, - 10, GETDATE()))**;**  **Go;** |

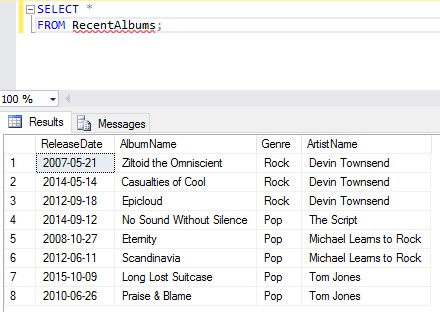
|  |
| --- |
| * In this example I added a semicolon to the end of the view. The Query Designer did not include this, but it's good practice to include it. * The semicolon is part of the ANSI SQL-92 standard. It is a statement terminator character. * Also, Microsoft has [announced](https://msdn.microsoft.com/en-us/library/ms143729.aspx) that non-semicolon ending Transact-SQL statements are deprecated in SQL Server 2019, and they won't be supported in a future version (SQL Server has historically used the keyword as a statement terminator **GO** instead of the semicolon). |

# Execute the View.



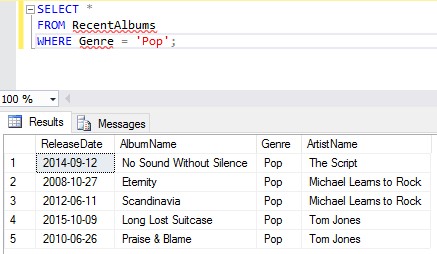
* Now execute the view just as you would execute any other query.
* Click Execute on the toolbar.
* You can now navigate to the view in the Object Explorer. Expand it and you will see the columns and their data types and properties — as though it was a table.

# Query the View



* Now that the view has been created, you can query the view by running a SELECT statement against it.
* So you can query our newly created view by using **SELECT \* FROM RecentAlbums**;.

# Filter the View



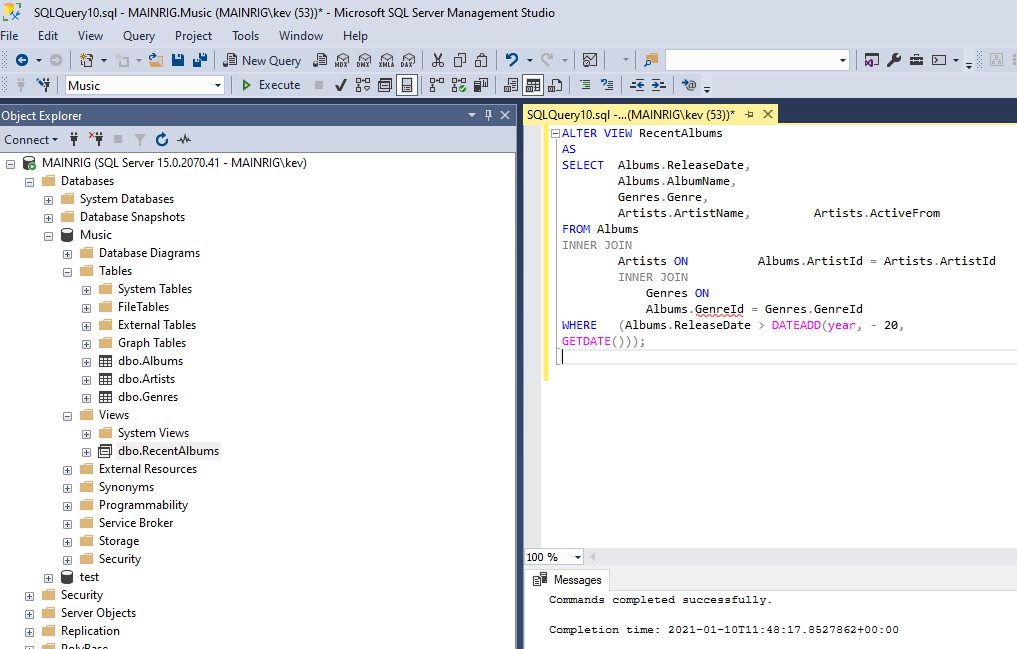
One of the good things about views is, you can apply your own filtering criteria against them — further filtering the results.

For example, you could add WHERE Genre = 'Pop', so that the view only returns *pop* albums from the last 10 years.

# Alter a View

You can modify your view by using the ALTER VIEW statement instead of the CREATE VIEW statement.

# Design the Altered View

Here we will modify our view to return albums over the past 20 years instead of just 10. 

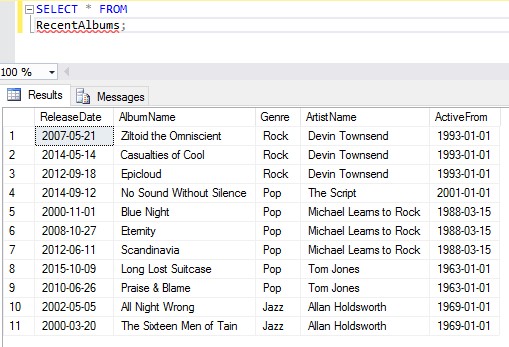
* It is a pretty old school collection, so any album released within the last 20 years is classified as "recent" :)
* We will also return another column: Artists.ActiveFrom

# Sample Code

Here's the code we use for the example – Copy and paste into a New Query and execute.

|  |
| --- |
| -- Task: (1) **Click** the **New Query** tab:  -- (2) **Copy** n **Paste** the below **TSQL** code into a new **Query Window.**  -- (3) Click on E**xecute** to run the **TSQL**: |
| ALTER VIEW RecentAlbums  AS  SELECT Albums.ReleaseDate,  Albums.AlbumName,  Genres.Genre,  Artists.ArtistName, Artists.ActiveFrom  FROM Albums  INNER JOIN  Artists ON Albums.ArtistId = Artists.ArtistId  INNER JOIN  Genres ON  Albums.GenreId = Genres.GenreId  WHERE (Albums.ReleaseDate > DATEADD(year, - 20,  GETDATE()));  **Go;** |

# Query the View



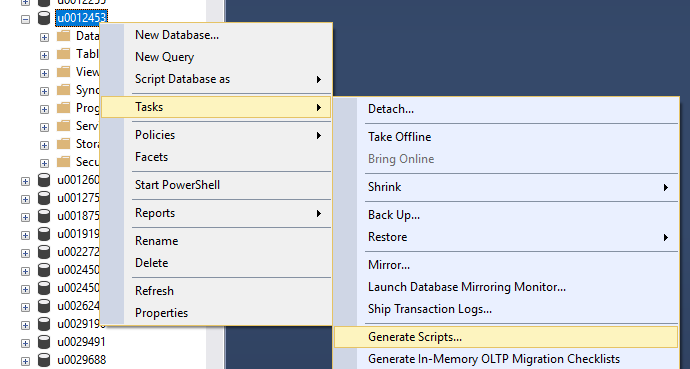
|  |
| --- |
| -- Task: (1) **Click** the **New Query** tab:  -- (2) **Copy** n **Paste** the below **TSQL** code into a new **Query Window.**  -- (3) Click on E**xecute** to run the **TSQL**: |
| SELECT \* from RecentAlbums;  Go; |

# Generating SQL Create / Insert Scripts Revisited

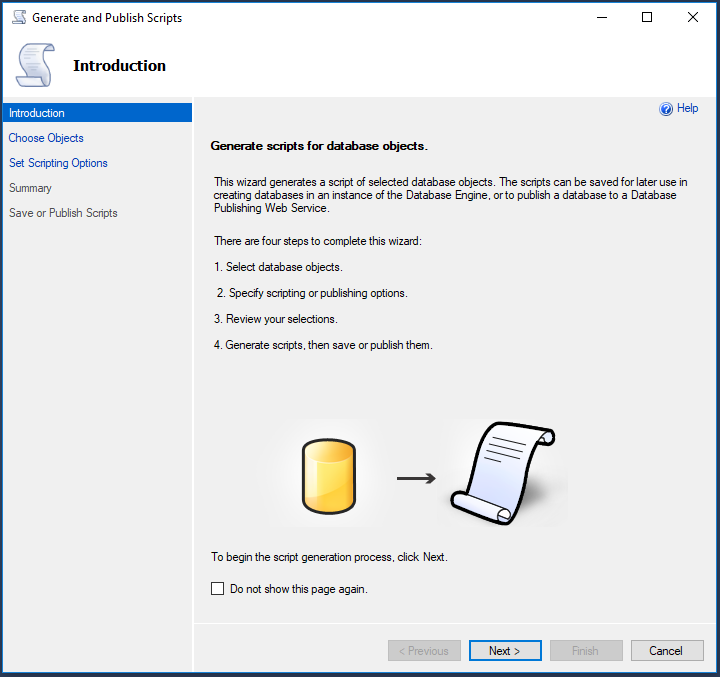
# Generating Database Script

To script a full database, expand Object Explorer and highlight the database name.

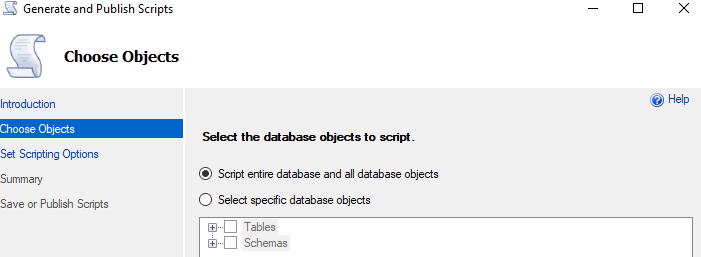
Right-click the database and then click Tasks>Generate Script:



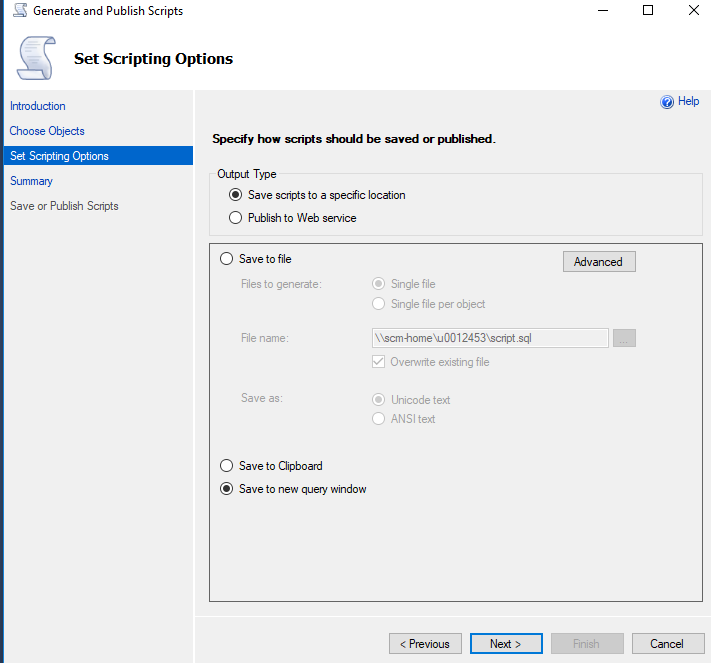
In the ‘Generate and Publish Scripts’ wizard that opens, click Next.



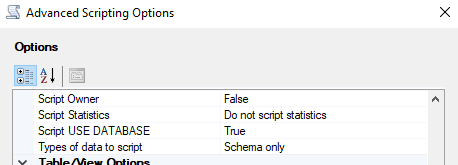
Select ‘Script entire database and all database objects’:



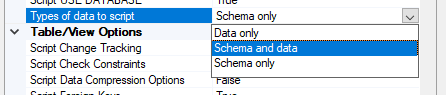
In the next screen, select ‘ Save to new query window’ and then click on the Advanced button:



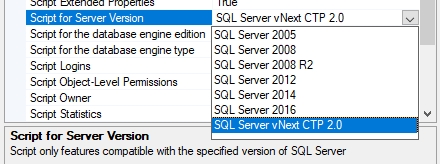
In the ‘Advanced Scripting Options’, scroll down to ‘Types of data to script’ and then mouse click next to ‘Schema only’



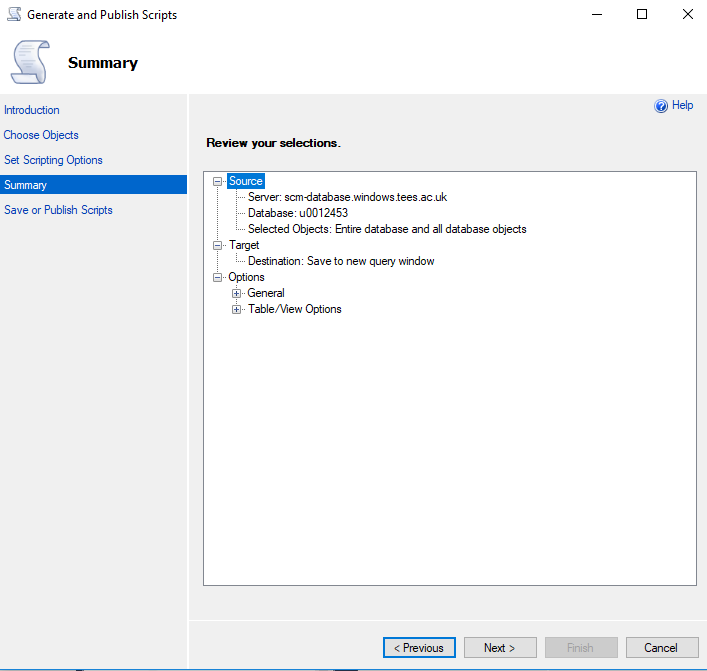
Open the dropdown and select ‘Schema and data’ and then click OK.



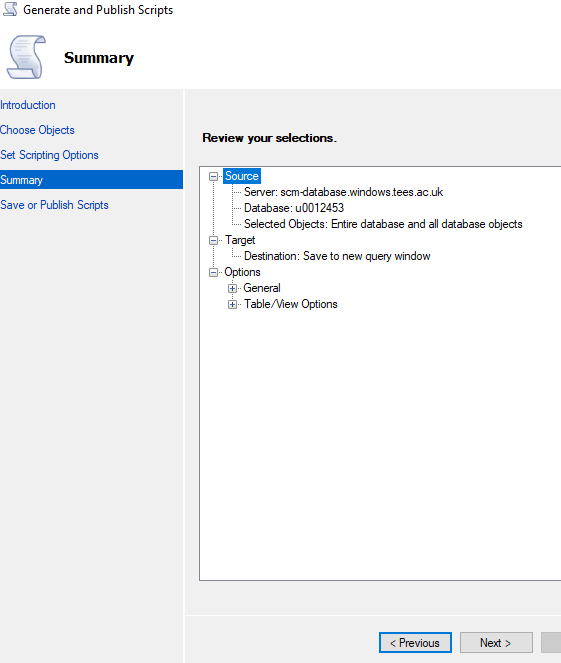
Note – You can also chose which Server Version to script for:



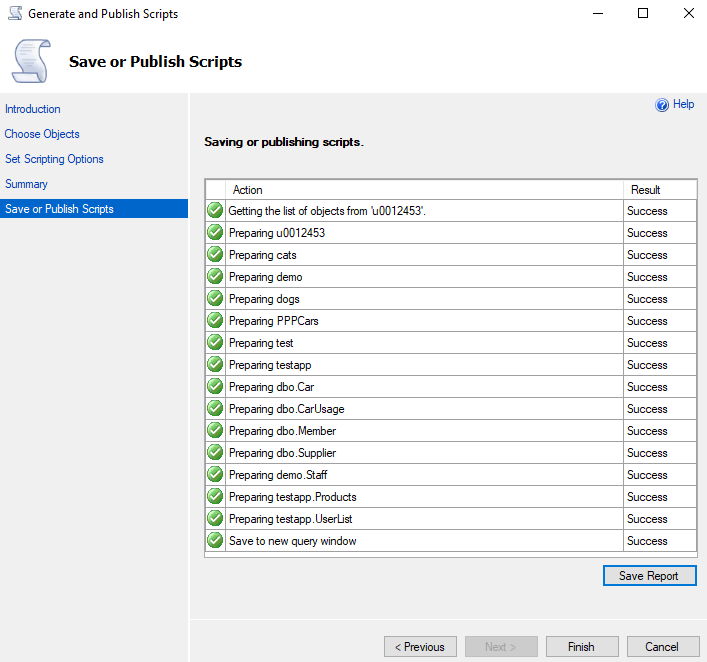
Back in the ‘Generate and Publish Scripts’ wizard click Next:



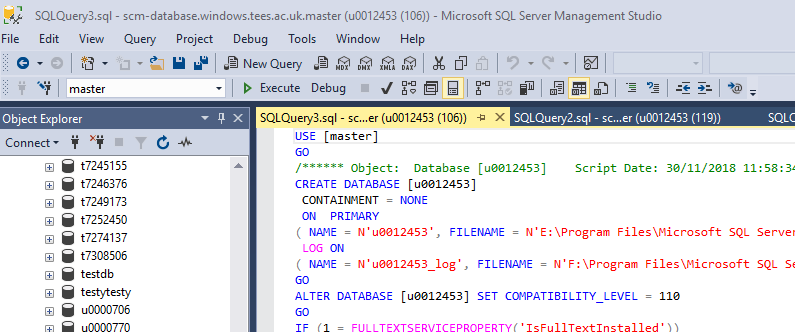
In the ‘Review your selections’ screen, click Next again:



You will then see the ‘Saving or publishing scripts’ screen - click Finish:



The generated script (to recreate the database and data) will now appear in the Query Editor window:

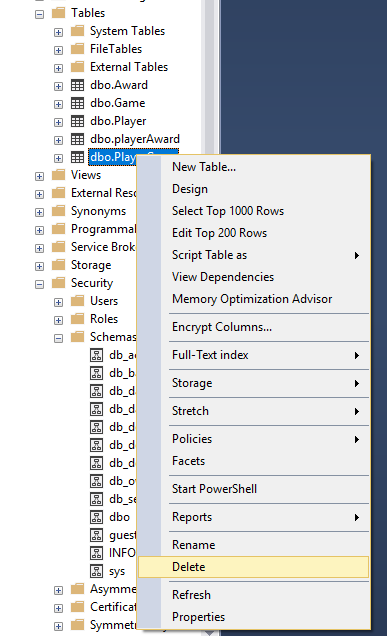
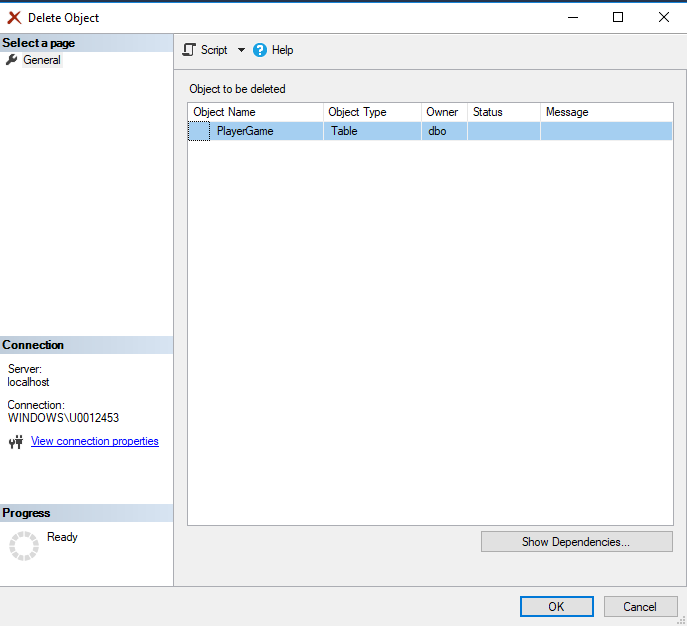


*Moving the database between home and Uni*

To recreate the latest version of your database (at home from the Uni version or at Uni from the home version) you will need to ‘drop’ (delete) the version that is already there –BUT make sure you have generated the script (as in the instructions above) to recreate the database first!

*Dropping the existing tables*

Right click on the last table in the list and select Delete then OK in the window that appears. Work up the list of tables deleting each one. Then repeat these actions for the Schema.

# Recreating the database at home – method 1

• Open your saved SQL Script file and find the very first Create Table command.

* **Delete every line before it and save the file.**
* Delete any older version of the Database if you have one and need to replace it with the later version.
* Alternatively create a new Database with a different version number - I recommend you keep 3 version numbers
* Double click your newly saved modified SQL Server Script file for automatic load into SQL SSMS Query Window or simply copy and past the revised code into a SQL Query Window.
* Ensure the Object Explorer is on the correct Database and execute the code. Remember to right click the Database and Refresh or press F5 key.
* Expand to view the Tables etc.
* Note I tend to run separate Create Scripts and Insert Scripts for larger Database.

# Recreating the database at home – method 2

* Open the generated sql code in the Query window in SSMS or in notepad.
* Find the first CREATE TABLE command and delete everything prior to this line.

At the bottom of the code, delete:

**USE [master]**

**GO**

**ALTER DATABASE [yourUserID] SET READ\_WRITE**

**GO**

Save the file.

* Expand Databases, right-click the database and click New Database. Give it a name.
* Confirm the correct database is selected, and then click OK.
* Click on New Query and copy/paste the revised code. Click Execute.
* Right click the database name and click refresh
* Expand the tables and you will see all the tables have been replicated along with the data if you also have the insert scripts

# Recreating the database at Uni

* Open the generated sql code in the Query window in SSMS or in notepad.
* Delete everything before this statement:

**USE [yourUserID]**

**GO**

**/\*\*\*\*\*\* Object: Schema [Tournament] Script Date: 07/12/2018 10:19:34 \*\*\*\*\*\*/**

**CREATE SCHEMA**

* At the bottom of the code, delete:

**USE [master]**

**GO**

**(This above the ALTER DATABASE [yourUserID] SET READ WRITE line).**

* Save the file.
* Click on New Query and copy/paste the revised code. Click Execute.
* Right click the database name and click refresh
* Expand the tables and you will see all the tables have been replicated along with the data if you also have the insert scripts

# Alternative method of loading and running the SQL generated Creates and Schemas