



Advanced SQL

Sample manual - first two chapters



Wise Owl
Training

TABLE OF CONTENTS (1 of 5)

1	THE MOVIES DATABASE	Page
1.1	Our Example Database	7

2	STORED PROCEDURES	Page
2.1	Overview	8
	<i>What is a Stored Procedure?</i>	8
	<i>Advantages and Disadvantages</i>	8
2.2	Creating Stored Procedures	9
	<i>Typing in a Stored Procedure</i>	9
	<i>Creating a Stored Procedure using a Template</i>	9
	<i>Executing the Query to Create your Stored Procedure</i>	10
	<i>Viewing your Stored Procedure</i>	10
2.3	Altering a Stored Procedure	11
	<i>Altering an Open Stored Procedure</i>	11
	<i>Altering a Procedure in a Database</i>	11
2.4	Executing Stored Procedures	12
	<i>Refreshing your Local Cache</i>	12
	<i>Executing a Procedure</i>	12
	<i>Altering and Executing a Stored Procedure Together</i>	13
	<i>Selecting a Stored Procedure Name to Run It</i>	13
2.5	Renaming and Deleting Stored Procedures	14
	<i>Renaming/Deleting a Procedure with the Menu</i>	14
	<i>Deleting a Procedure in Script</i>	14
	<i>Renaming a Procedure in Script</i>	14
2.6	System Stored Procedures	15
	<i>Listing System Stored Procedures</i>	15
	<i>Useful System Stored Procedures</i>	16
2.7	Getting Help on SQL	17
	<i>Context-Sensitive Help</i>	17
	<i>Tips on Googling</i>	17

3	VARIABLES	Page
3.1	Declaring Variables	18
	<i>Syntax for Declaring a Variable</i>	18
3.2	Using Variables	19
	<i>Setting the Value of a Variable</i>	19
	<i>Showing the Values of Variables</i>	19
	<i>Scope of Variables</i>	20
	<i>Incrementing and Concatenating Variables</i>	20
	<i>The Importance of Casting</i>	21
3.3	Using Variables with Subqueries	22
	<i>An Alternative Approach: Aggregate Functions</i>	22
3.4	Storing Column Values in Variables	23
	<i>Storing a Single Row's Values</i>	23
	<i>Accumulating Numbers</i>	23
	<i>Accumulating Text</i>	24
3.5	Global Variables	25
	<i>Special Considerations when using @@ROWCOUNT</i>	25

4	VARIABLE AND PARAMETER DATA TYPES	Page
4.1	Numeric Data Types	26
	<i>Integer Variable/Parameter Types</i>	26
	<i>Decimal and Numeric Types</i>	26
4.2	Character Data Types	27
	<i>Types of Character Storage</i>	27
	<i>Variable Length Data Types</i>	27
	<i>Fixed Length Data Types</i>	27
4.3	Date/Time Data Types	28

TABLE OF CONTENTS (2 of 5)

5	STORED PROCEDURE PARAMETERS	Page
5.1	Overview	29
	<i>Syntax of Parameters</i>	29
5.2	Simple Parameters	30
	<i>Step 1 – Specifying the Parameters</i>	30
	<i>Step 2 – Coding the Parameters</i>	30
	<i>Step 3 – Referencing the Parameters</i>	31
	<i>Using Text Wildcards as Parameters</i>	31
5.3	Running Procedures using Parameters	32
	<i>Positional Arguments</i>	32
	<i>Named Arguments</i>	33
	<i>Right-clicking to Execute a Procedure</i>	33
5.4	Default Parameter Values	34
	<i>Setting Default Values to Null</i>	35
	<i>The Perfect Stored Procedure?</i>	35
5.5	The RETURN Statement	36
5.6	Output Parameters	37

6	CONDITIONS AND LOOPS	Page
6.1	IF Conditions	38
	<i>Simple Conditions</i>	38
	<i>Using BEGIN ... END</i>	38
	<i>Using ELSE</i>	39
	<i>Nesting Conditions and Indentation</i>	39
	<i>Using CASE to Avoid IF</i>	40
6.2	Looping using WHILE	41
	<i>The Syntax of WHILE Loops</i>	41
	<i>Breaking out of Loops</i>	42

7	SCALAR FUNCTIONS	Page
7.1	Overview	43
	<i>Syntax of a Scalar Function</i>	43
7.2	Writing a Scalar Function	44
	<i>Specifying Input Parameters and Return Types</i>	44
	<i>Writing the Function Itself</i>	44
7.3	Running a Function	45
	<i>Calling a Function on its Own</i>	45
	<i>Calling a Function within a SELECT Statement</i>	45
7.4	Worked Examples	46
	<i>Example One – Returning a Person's Status</i>	46
	<i>Example Two – Profitability</i>	47
	<i>Example Three – Categorisation (by Oscar Type)</i>	48
7.5	Limitations of Functions	49
	<i>Assessing Function Speed</i>	49

8	ERROR TRAPPING	Page
8.1	About Errors	50
8.2	TRY / CATCH	51
	<i>Syntax of TRY / CATCH</i>	51
	<i>Example of a Simple Error Trap</i>	51
	<i>Nesting TRY Statements</i>	52
8.3	Error Functions	53
	<i>T-SQL Error Functions</i>	53
	<i>Error Severity Levels</i>	54
	<i>Showing Errors within a TRY / CATCH Block</i>	54
8.4	Customising Error Messages	55
	<i>Viewing the Full List of Error Messages</i>	55
	<i>Creating your Own Errors</i>	56
	<i>Customising your own Error Messages</i>	56

9	DELETING DATA	Page
9.1	Deleting (Dropping) Tables	57
	<i>Dropping a Table if it Exists</i>	57
	<i>Using Error Trapping to Check Existence</i>	57
	<i>Using SYS.OBJECTS and OBJECT_ID</i>	58
9.2	Deleting Rows	59
	<i>Differences between TRUNCATE and DELETE FROM</i>	59

TABLE OF CONTENTS (3 of 5)

10	UPDATING DATA	Page
10.1	The UPDATE Command	60
	<i>An Example – Changing Genres for Films</i>	60
10.2	Updating using JOIN	61
	<i>The Obvious Answer doesn't Work</i>	61
	<i>The Correct Syntax</i>	62

11	INSERTING DATA	Page
11.1	Three Possible Ways to Insert	63
11.2	Creating Tables from Existing Data (SELECT INTO)	64
	<i>Step 1 – Getting the Data for your New Table</i>	64
	<i>Step 2 – Making a New Table</i>	64
	<i>Step 3 – Checking the Table Created</i>	65
11.3	Inserting Multiple Rows into an Existing Table	66
	<i>Step 1 – Understanding the Syntax</i>	66
	<i>Step 2 – Working out what to do</i>	66
	<i>Step 3 – Mapping the Columns</i>	67
	<i>Step 4 – Creating the Query</i>	67
11.4	Inserting Single Rows	68
	<i>Syntax of INSERT INTO ... VALUES</i>	68
	<i>Example Code to Insert a New Row</i>	68
	<i>Inserting a Batch of Single Rows</i>	69
11.5	INSERT INTO – More Possibilities	70
	<i>Missing out Columns</i>	70
	<i>Using a Stored Procedure's Output</i>	70
	<i>Outputting Inserted Rows</i>	71
	<i>Getting Inserted Row Numbers with @@IDENTITY</i>	71

12	CREATING TABLES	Page
12.1	Setting Up our Example	72
	<i>The Example Used in this Chapter</i>	72
	<i>Creating and Dropping Databases</i>	72
12.2	Creating Tables	73
12.3	Setting Primary Keys	74
	<i>Creating a Primary Key when Creating Tables</i>	74
	<i>Creating a Primary Key Afterwards</i>	74
12.4	Setting a Default Value for a Column	75
12.5	Preventing Null Values in a Column	76
12.6	Putting Checks or Constraints on a Column	77
12.7	Foreign Keys and Relationships	78
	<i>Our Example</i>	78
	<i>Foreign Keys</i>	79
	<i>Creating a Foreign Key Constraint</i>	79
12.8	Two Reasons/Ways to Index a Column	80
	<i>Creating an Index to Speed Up Queries</i>	80
	<i>Enforcing Uniqueness with an Index</i>	80
12.9	A Complete Example	81

13	TRANSACTIONS	Page
13.1	The Concept	82
	<i>Syntax of a Transaction</i>	82
13.2	A Simple Example	83
13.3	Case Study – Recategorising Films	84
	<i>The Problem</i>	84
	<i>The Algorithm</i>	84
	<i>The Procedure</i>	85
13.4	Errors and Transactions	86

TABLE OF CONTENTS (4 of 5)

14	TEMPORARY TABLES	Page
14.1	Overview of Temporary Tables	87
	<i>Local and Global Temporary Tables</i>	87
	<i>How Temporary Tables are Stored</i>	87
14.2	Creating and Deleting Temporary Tables	88
	<i>Creating Temporary Tables</i>	88
	<i>Deleting Temporary Tables</i>	88
14.3	Scope of Temporary Tables	89
	<i>Temporary Tables are Tied to the Queries Creating Them</i>	89
	<i>Visibility of Temporary Tables</i>	90
	<i>Scope of Temporary Tables in Stored Procedures</i>	91
14.4	Case Study – Successful People	92
	<i>Step 1 – Busy Actors (Creating the Table)</i>	93
	<i>Step 2 – Busy Directors (Inserting Rows)</i>	93
	<i>Final Answer with Problems Solved</i>	94

15	TABLE VARIABLES	Page
15.1	About Table Variables	95
15.2	Case Study Revisited	96

16	COMPARING TABLE TYPES	Page
16.1	Differences between Table Variables and Temporary Tables	97
	<i>Speed</i>	97
	<i>Limitations of Table Variables</i>	98
	<i>Limitations of Temporary Tables</i>	98

17	TABLE-VALUED FUNCTIONS	Page
17.1	The Two Types of Table-Valued Functions	99
	<i>Types of Table-Valued Functions</i>	99
	<i>Where to Find Them</i>	99
17.2	In-line Table-Valued Functions	100
	<i>Syntax of In-Line TVFs</i>	100
	<i>Where Stored Procedures Fall Short</i>	100
	<i>The In-Line TVF Solution</i>	101
	<i>Joins with Table-Valued Functions</i>	101
17.3	Multi-Statement Table-Valued Functions	102
	<i>Syntax of an MSTVF</i>	102
	<i>Example of an MSTVF</i>	103

18	CURSORS	Page
18.1	About Cursors	104
	<i>Reasons to Use Cursors</i>	104
	<i>The Syntax of a Basic Cursor</i>	104
18.2	Example of a Cursor	105

19	DEBUGGING IN SQL	Page
19.1	Example Used	106
19.2	Debugging	107
	<i>Starting and Stopping Debugging</i>	107
	<i>Stepping Through Code</i>	108
	<i>Setting and Unsetting Breakpoints</i>	108
	<i>Viewing Variable Values by Hovering</i>	109
	<i>Viewing Variables in the Locals Window</i>	109

20	DYNAMIC SQL	Page
20.1	The EXEC Command and Dynamic SQL	110
	<i>Why not to Use Dynamic SQL</i>	110
20.2	Example –Parameterising Row Selection	111

21	CTES AND DERIVED TABLES	Page
21.1	Multi-Stage Queries	112
21.2	Derived Tables	113
21.3	Single CTEs (Common Table Expressions)	114
	<i>Syntax of Single CTEs</i>	114
	<i>The CTE for our Example</i>	114
21.4	Multiple CTEs	115
	<i>Syntax of Multiple CTEs</i>	115
	<i>Example of a Multiple CTE</i>	116

22	SUBQUERIES	Page
22.1	Single-Value Subqueries	117
	<i>Example: Showing the Name of the Longest Film</i>	117
22.2	ANY, ALL, IN and NOT IN	118
22.3	Correlated Subqueries	119
	<i>Correlated Subqueries: Definition and Example</i>	119
	<i>Alternatives to Correlated Subqueries</i>	119
	<i>Considering Speed</i>	120
	<i>Using EXISTS to Check whether Rows are Returned</i>	120

TABLE OF CONTENTS (5 of 5)

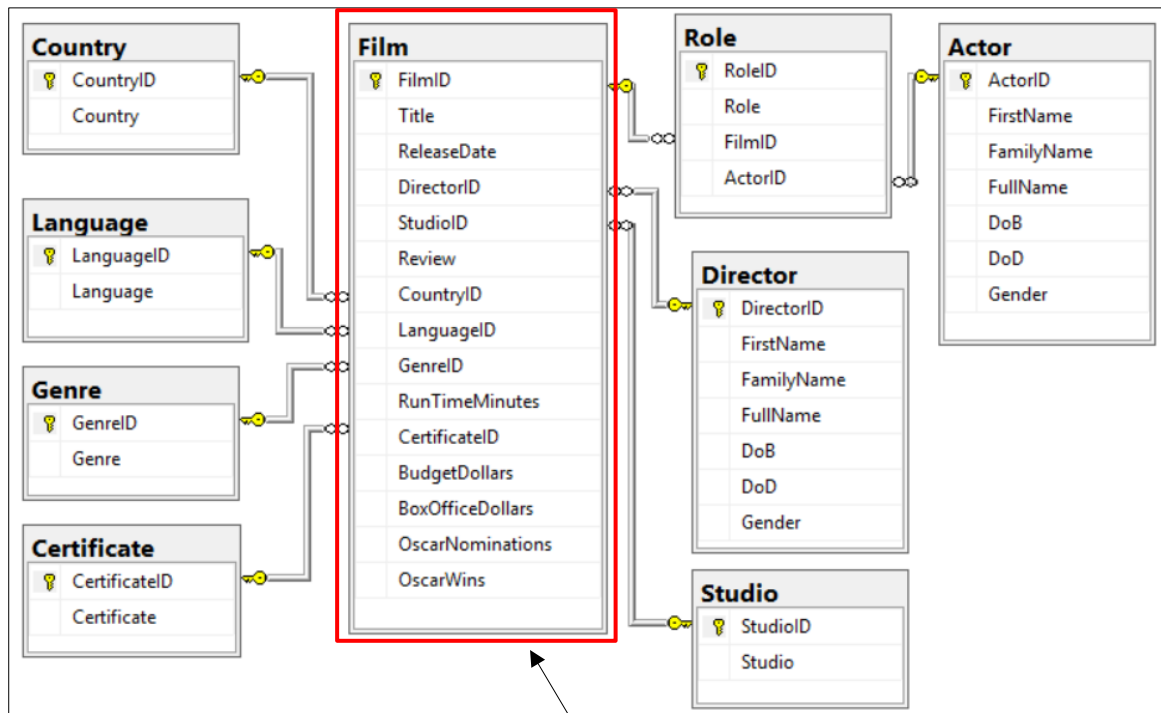
23	PIVOTING DATA	Page
23.1	Overview	121
23.2	The Two Stages of Creating a Pivot Query	122
	<i>Step 1 – Assembling the Data</i>	122
	<i>Step 2 – Pivoting the Assembled Data</i>	123
23.3	Varying the Number of Row Fields	124
	<i>Pivot Queries with no Row Headings</i>	124
	<i>Pivot Queries with Multiple Row Headings</i>	125
23.4	Queries Based on Pivot Queries	126
23.5	Getting and Using Dynamic Columns	127
	<i>Step 1 – Get a Comma-Delimited List</i>	127
	<i>Step 2 – Build up the SQL Statement</i>	128
	<i>Step 3 – Test the SQL</i>	128
	<i>Step 4 – Execute the SQL</i>	128

24	TRIGGERS	Page
24.1	Overview of Triggers	129
	<i>Syntax of a Trigger</i>	129
24.2	Working with Triggers	130
	<i>Creating a Trigger</i>	130
	<i>Viewing Triggers</i>	130
	<i>Enabling and Disabling Triggers</i>	131
	<i>Deleting Triggers</i>	131
24.3	More Sophisticated Triggers	132
	<i>Tables Created by Triggers</i>	132
24.4	A Case Study: Transactions in Triggers	133

CHAPTER 1 - THE MOVIES DATABASE

1.1 Our Example Database

The database used throughout this manual contains 1,200 films, with associated details:



For each film there is an associated country, language, genre, certificate, director and studio. In addition there is a table of actors, and a **Role** table which links films and actors together (as explained below).

The **Role** table looks like this:

Each row contains the details of one actor who played a role in a film.

RoleID	Role	FilmID	ActorID
1	Ray Ferrier	33	1
2	Dr. Alan Grant	1	2
3	Dr. Ellie Sattler	1	3
4	Dr. Ian Malcolm	1	4

```
-- who played which roles
SELECT
  f.Title AS 'Film',
  a.FullName AS 'Actor',
  r.Role AS 'Role'
FROM
  Film AS f
  INNER JOIN Role AS r
    ON f.FilmID = r.FilmID
  INNER JOIN Actor AS a ON
    r.ActorID = a.ActorID
ORDER BY
  r.RoleID
```

You could run a query like this to list out all of the roles in the database, with who played this part and in which film:

Film	Actor	Role
War of the Worlds	Tom Cruise	Ray Ferrier
Jurassic Park	Sam Neill	Dr. Alan Grant
Jurassic Park	Laura Dern	Dr. Ellie Sattler
Jurassic Park	Jeff Goldblum	Dr. Ian Malcolm

CHAPTER 2 - STORED PROCEDURES

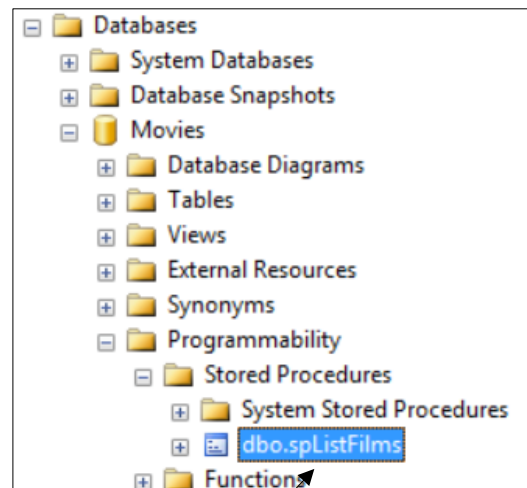
2.1 Overview

What is a Stored Procedure?

A stored procedure is a set of SQL instructions (often just a single **SELECT** statement) which is saved within your database:

```
CREATE PROC spListFilms
AS
-- list out all of the films
SELECT
  f.Title
  , f.OscarWins AS Oscars
  , f.RunTimeMinutes AS RunTime
FROM
  Film AS f
```

This is the code to create a stored procedure, here called **spListFilms**.



Here is the created procedure, in the **Programmability** section of your database.

Advantages and Disadvantages

Stored procedures have many advantages:

Advantage	Notes
<i>Range of commands</i>	Whereas a query can only select data, a stored procedure can also insert, update and delete rows (not to mention creating and dropping tables).
<i>Debugging</i>	You can step through a stored procedure line by line to see what it's doing (although this strangely isn't that useful).
<i>Parameters</i>	Above all, you can pass parameters to a stored procedure (although we won't do this until a later chapter). For example, you could write a procedure to list all the films made between any two given dates, winning at least N Oscars.

Against all this is one potential disadvantage: because stored procedures are so powerful, not all IT departments are that keen on giving people the authority to create and execute them!



One common misconception about stored procedures is that they run faster than simple queries. They don't, since SQL Server will create an optimised execution plan in either case.

2.2 Creating Stored Procedures

Typing in a Stored Procedure

The best way to create a procedure is to press **Ctrl** + **N** to create a query, then use this syntax:

a) Begin a stored procedure with **CREATE PROC** then give your procedure a name.

b) Although it serves no purpose, you need the keyword **AS** to separate the instruction to create a procedure from what it does.

c) Finally, you need to say what your stored procedure does. This can run to hundreds of lines of code – creating tables and manipulating data – but for this chapter we'll just stick to selecting a set of rows from a table.

```

USE Movies
GO

-- this must be the first statement
-- in the batch (we needed that GO)
CREATE PROC spListFilms

-- the word AS is a necessary but
-- meaningless link word
AS

-- list out all of the films
SELECT
    f.Title
    , f.OscarWins AS Oscars
    , f.RunTimeMinutes AS RunTime
FROM
    Film AS f

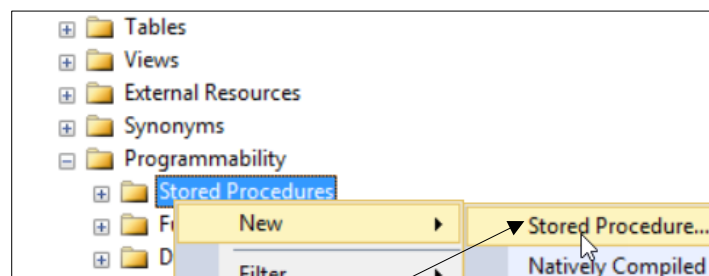
```



*It's a common convention to begin procedure names with **sp**, as above. However, avoid using **sp_** as a prefix, since this is reserved for system stored procedures (and Microsoft may create one in the future which clashes with your name!).*

Creating a Stored Procedure using a Template

This is Microsoft trying to be helpful, but failing!



a) Expand your database to choose **Programmability** → **Stored Procedures** → **New** → **Stored Procedure ...**

b) Edit this template. The trouble is that while you're a stored procedure newbie it's more baffling than helpful, and when you know how to create procedures it's quicker to type them in yourself!

```

Command (Ctrl+Shift+N) to fill in the parameter
-- values below.
--
-- This block of comments will not be included in
-- the definition of the procedure.
-----
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
-----
-- Author:      <Author,,Name>
-- Create date: <Create Date,,>
-- Description: <Description,,>
-----
CREATE PROCEDURE <Procedure_Name, sysname, Procedure
-- Add the parameters for the stored procedure here
<@Param1, sysname, @p1> <Datatype_For_Param1, ,
<@Param2, sysname, @p2> <Datatype_For_Param2, ,
AS
BEGIN
-- SET NOCOUNT ON added to prevent extra result
-- interfering with SELECT statements.
SET NOCOUNT ON;

-- Insert statements for procedure here
SELECT <@Param1, sysname, @p1>, <@Param2, sysname
END

```

Executing the Query to Create your Stored Procedure

Once you've typed in SQL to create a stored procedure, it's time to run this:

```

CREATE PROC spListFilms
AS
-- list out all of the films
SELECT
    f.Title
    , f.OscarWins AS Oscars
    , f.RunTimeMinutes AS RunTime
FROM
    Film AS f
  
```

8 %

Messages
Command(s) completed successfully.

a) As for a query, click anywhere in the stored procedure and press **F5** to execute it.

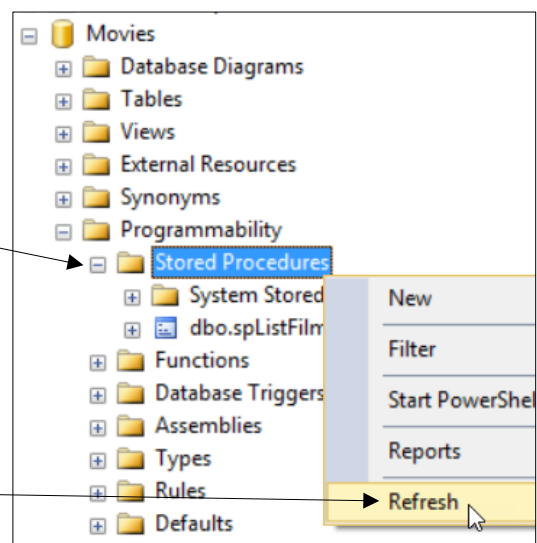
b) If all goes well (and you haven't made any mistakes) you'll see this message, to show you that the stored procedure has been successfully created.

Viewing your Stored Procedure

To check SSMS has created your stored procedure, expand your database as shown here:

a) In the **Programmability** section, you should be able to expand **Stored Procedures** to see the one you've created.

b) If you can't see the procedure you've just created, right-click on **Stored Procedures** and choose **Refresh** as shown here to bring the list up to date.



*If you still can't see your stored procedure, by far the most likely reason is that you've created it in one database (probably the **master** one), but are looking in another!*

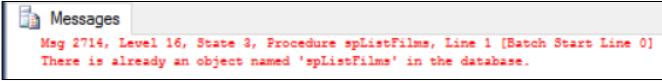
2.3 Altering a Stored Procedure

If you want to change what a stored procedure does, in the strange world of Management Studio you need to write *script* to alter it.

Altering an Open Stored Procedure

If you've just been working with a stored procedure, it's easy to change it:

a) Change the word **CREATE** to **ALTER**. If you don't do this, you'll see this message when you run your script:



b) Make any other changes to your procedure (here we've tacked on an **ORDER BY** clause to sort the films by title).

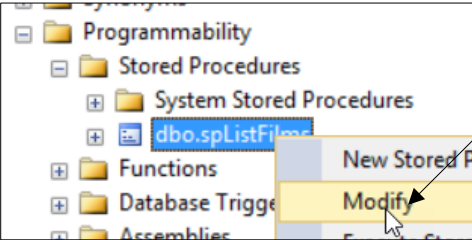
```
ALTER PROC spListFilms
AS
-- list out all of the films
SELECT
    f.Title
    , f.OscarWins AS Oscars
    , f.RunTimeMinutes AS RunTime
FROM
    Film AS f
ORDER BY
    f.Title
```

When you run the script you'll again see the message **Command(s) completed successfully**. This means SSMS has deleted the old version of your procedure and replaced it with your new one.

Altering a Procedure in a Database

If your procedure isn't open, follow these steps to make changes to it (you can then execute the script to change what the procedure does, as shown above):

a) Right-click on the procedure that you want to change, and choose to modify it. SSMS will generate a new query containing the script shown below.



b) Although you don't have to, it's a good idea to delete these added lines of SQL to remove clutter. Here's what they do, and why you won't miss them:

Line	Notes
USE [Movies]	You're already using this database!
SET ANSI_NULLS ON	Obscure changes to the way nulls and quotation marks are treated, which are of no consequence or relevance.
SET QUOTED_IDENTIFIER ON	

If you find this explanation a bit lacking, go to <http://bit.ly/2kQ1mfx> for more details (but you're not missing anything, honest!).

```
USE [Movies]
GO
/***** Object: StoredProcedure [
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
ALTER PROC [dbo].[spListFilms]
AS
-- list out all of the films
SELECT
    f.Title
    , f.OscarWins AS Oscars
    , f.RunTimeMinutes AS RunTime
FROM
    Film AS f
```

2.4 Executing Stored Procedures

Before running a procedure, it's first a good idea to persuade SSMS your procedure exists!

Refreshing your Local Cache

You can (as we'll see in a moment) run a stored procedure using the **EXEC** command, but you have to persuade Management Studio that your stored procedure actually exists:

IntelliSense doesn't know what you're talking about ...

... and when you type in the name of the procedure SSMS underlines it in red (although it shows as an error, this command will actually run).

The easy way to get SSMS to acknowledge your new procedure exists is to update its memory of what's in your database. To do this select: **Edit** → **IntelliSense** → **Refresh Local Cache** .

Wise Owl's Hint

However, it's much easier just to press **Ctrl** + **Shift** + **R** .

Executing a Procedure

The commands shown here would run your procedure:

Each of these commands would run a procedure called **spListFilms**.

The output from these two commands: SSMS will run the procedure twice, and hence show two sets of output.

	Title	Oscars	RunTime
1	10	0	122
2	101 Dalmatians	0	103
3	12 Years a Slave	3	134
4	127 Hours	0	93
5	13 Assassins	0	125

Query executed successfully.

The **GO** above is vital, otherwise SSMS will read the command as this:

Without the **GO** Management Studio would run the two commands together, and shown this error message:

```
Msg 8146, Level 16, State 1, Procedure spListFilms, Line 0 (Batch Start Line 2)
Procedure spListFilms has no parameters and arguments were supplied.
```

-- run a stored procedure with
-- a parameter
EXEC spListFilms spListFilms

Altering and Executing a Stored Procedure Together

A common way to run a procedure is immediately after creating or changing it:

This part of the script alters the existing procedure, replacing whatever it used to do with new code. Don't worry if the old code and the new code are actually exactly the same!

After finishing the previous batch of statements, to modify what the stored procedure does, we now execute it.

```
ALTER PROC spListFilms
AS
-- list out all of the films
SELECT
    f.Title
    , f.OscarWins AS Oscars
    , f.RunTimeMinutes AS RunTime
FROM
    Film AS f
ORDER BY
    f.Title
-- finish creating or altering
-- the procedure!
GO
-- NOW we can run it
spListFilms
```



You need the **GO** above because otherwise you would create a script which tried to run itself, which SSMS wouldn't be happy with!

Selecting a Stored Procedure Name to Run It

For a simple stored procedure (one which you can run without specifying any parameters), the easiest way to run it is often just to select it and press **F5**.

```
ALTER PROC spListFilms
AS
-- list out all of the
films
SELECT
    f.Title
```

a) Double-click on the name of the procedure to select it, then press **F5**.

b) SSMS will run your procedure and show its output.

	Title	Oscars	RunTime
1	10	0	122
2	101 Dalmatians	0	103
3	12 Years a Slave	3	134
4	127 Hours	0	93
5	13 Assassins	0	125

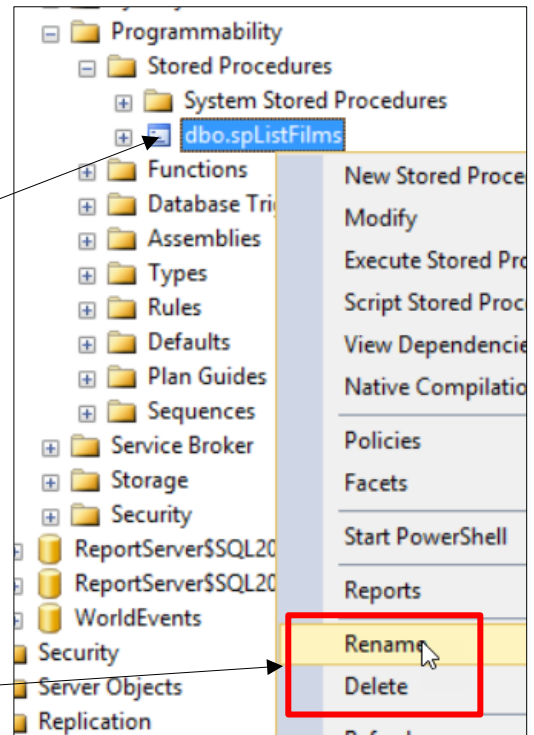
2.5 Renaming and Deleting Stored Procedures

Renaming/Deleting a Procedure with the Menu

To change the name of a stored procedure, or delete it, right-click on it:

a) Find the procedure that you want to rename or delete, and right-click on it.

b) Choose one of these options to either change its name or delete it.



Deleting a Procedure in Script

To delete a procedure, you *drop* it:

```
-- delete a procedure
DROP PROC spListFilms
```

Run this command to permanently delete the stored procedure called **spListFilms**.

Renaming a Procedure in Script

To change the name of a procedure in script, create a new version with the new name and then delete the old one:

a) Create script to modify the procedure as shown in the previous pages.

```
ALTER PROC spListFilms
AS
-- list out all films
SELECT
    f.Title
```

b) Change **ALTER** to **CREATE**, type in a new name for the procedure then execute this.

```
CREATE PROC spNewName
AS
-- list out all films
SELECT
    f.Title
```

c) Change the command to drop the original procedure,

```
DROP PROC
    spListFilms
```

2.6 System Stored Procedures

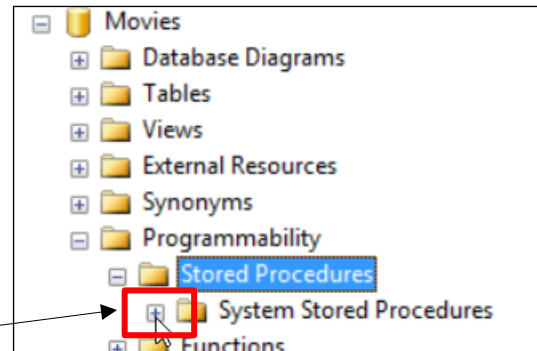
Listing System Stored Procedures

SQL Server comes with many built-in system stored procedures (1,390 in the version being used to write this courseware). Here are two ways to show these:

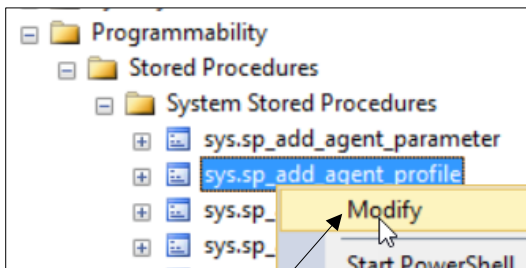
```
-- list system stored procedures
SELECT o.Name AS 'Procedure'
FROM sys.system_objects as o
WHERE o.type = 'P'
ORDER BY o.Name
```

Either run this script to show all the system objects which are procedures ...

... or click on the **+** symbol to list them all.



You can then choose to look at any of them, although you may regret it ...



You can right-click on any of the system stored procedures to change them ...

```
/*
** The system profile of the same type of agent
** the parameters in this new user profile.
*/
ALTER procedure [sys].[sp_add_agent_profile] (
    @profile_id          int = NULL OUTPUT,
    @profile_name        sysname,
    @agent_type          int,
    --
```

... but the contents won't be easy to read (Wise Owl have absolutely no idea what this procedure does, for example!).

Useful System Stored Procedures

Here are some stored procedures which you might like to try:

Procedure	What it does	Example results																																	
sp_help	Lists out all of the tables, views, etc in your database (you can also press Alt + F1 to do this).	<table border="1"> <thead> <tr> <th>Name</th> <th>Owner</th> <th>Object_type</th> </tr> </thead> <tbody> <tr> <td>View_1</td> <td>dbo</td> <td>view</td> </tr> <tr> <td>Actor</td> <td>dbo</td> <td>user table</td> </tr> <tr> <td>Certificate</td> <td>dbo</td> <td>user table</td> </tr> <tr> <td>Country</td> <td>dbo</td> <td>user table</td> </tr> <tr> <td>Director</td> <td>dbo</td> <td>user table</td> </tr> </tbody> </table>	Name	Owner	Object_type	View_1	dbo	view	Actor	dbo	user table	Certificate	dbo	user table	Country	dbo	user table	Director	dbo	user table															
Name	Owner	Object_type																																	
View_1	dbo	view																																	
Actor	dbo	user table																																	
Certificate	dbo	user table																																	
Country	dbo	user table																																	
Director	dbo	user table																																	
sp_help 'Table'	Lists out all the details of (and columns in) any specified table (eg sp_help 'Film')	<table border="1"> <thead> <tr> <th>Name</th> <th>Owner</th> <th>Type</th> <th>Created_datetime</th> </tr> </thead> <tbody> <tr> <td>Film</td> <td>dbo</td> <td>user table</td> <td>2017-01-26 10:58:06.243</td> </tr> </tbody> </table> <table border="1"> <thead> <tr> <th>Column_name</th> <th>Type</th> <th>Computed</th> <th>Length</th> <th>Prec</th> </tr> </thead> <tbody> <tr> <td>FilmID</td> <td>int</td> <td>no</td> <td>4</td> <td>10</td> </tr> <tr> <td>Title</td> <td>nvarchar</td> <td>no</td> <td>510</td> <td></td> </tr> <tr> <td>ReleaseDate</td> <td>datetime</td> <td>no</td> <td>8</td> <td></td> </tr> <tr> <td>DirectorID</td> <td>int</td> <td>no</td> <td>4</td> <td>10</td> </tr> </tbody> </table>	Name	Owner	Type	Created_datetime	Film	dbo	user table	2017-01-26 10:58:06.243	Column_name	Type	Computed	Length	Prec	FilmID	int	no	4	10	Title	nvarchar	no	510		ReleaseDate	datetime	no	8		DirectorID	int	no	4	10
Name	Owner	Type	Created_datetime																																
Film	dbo	user table	2017-01-26 10:58:06.243																																
Column_name	Type	Computed	Length	Prec																															
FilmID	int	no	4	10																															
Title	nvarchar	no	510																																
ReleaseDate	datetime	no	8																																
DirectorID	int	no	4	10																															
sp_columns 'Table'	Another way to list all the columns included in a particular table (eg sp_columns 'Director').	<table border="1"> <thead> <tr> <th>TABLE_NAME</th> <th>COLUMN_NAME</th> <th>DATA_TYPE</th> <th>TYPE_ID</th> </tr> </thead> <tbody> <tr> <td>Director</td> <td>DirectorID</td> <td>4</td> <td>int iden</td> </tr> <tr> <td>Director</td> <td>FirstName</td> <td>-9</td> <td>nvarchar</td> </tr> <tr> <td>Director</td> <td>FamilyName</td> <td>-9</td> <td>nvarchar</td> </tr> <tr> <td>Director</td> <td>FullName</td> <td>-9</td> <td>nvarchar</td> </tr> <tr> <td>Director</td> <td>DoB</td> <td>11</td> <td>datetime</td> </tr> </tbody> </table>	TABLE_NAME	COLUMN_NAME	DATA_TYPE	TYPE_ID	Director	DirectorID	4	int iden	Director	FirstName	-9	nvarchar	Director	FamilyName	-9	nvarchar	Director	FullName	-9	nvarchar	Director	DoB	11	datetime									
TABLE_NAME	COLUMN_NAME	DATA_TYPE	TYPE_ID																																
Director	DirectorID	4	int iden																																
Director	FirstName	-9	nvarchar																																
Director	FamilyName	-9	nvarchar																																
Director	FullName	-9	nvarchar																																
Director	DoB	11	datetime																																
sp_helptext 'Procedure'	Returns the lines in a stored procedure, view or function as a table (what you do with this is not obvious!).	<table border="1"> <thead> <tr> <th>Text</th> </tr> </thead> <tbody> <tr> <td>1 CREATE PROC spExample</td> </tr> <tr> <td>2 AS</td> </tr> <tr> <td>3</td> </tr> <tr> <td>4 -- list out all films</td> </tr> <tr> <td>5 SELECT</td> </tr> <tr> <td>6 f.Title</td> </tr> <tr> <td>7 , f.OscarWins AS Oscars</td> </tr> </tbody> </table>	Text	1 CREATE PROC spExample	2 AS	3	4 -- list out all films	5 SELECT	6 f.Title	7 , f.OscarWins AS Oscars																									
Text																																			
1 CREATE PROC spExample																																			
2 AS																																			
3																																			
4 -- list out all films																																			
5 SELECT																																			
6 f.Title																																			
7 , f.OscarWins AS Oscars																																			
sp_datatype_info	Shows information on the data types in SQL, to jog your memory.	<table border="1"> <thead> <tr> <th>TYPE_NAME</th> <th>DATA_TYPE</th> <th>PRECISION</th> <th>LITERAL</th> </tr> </thead> <tbody> <tr> <td>sql_variant</td> <td>-150</td> <td>8000</td> <td>NULL</td> </tr> <tr> <td>uniqueidentifier</td> <td>-11</td> <td>36</td> <td>'</td> </tr> <tr> <td>ntext</td> <td>-10</td> <td>1073741823</td> <td>N'</td> </tr> <tr> <td>xml</td> <td>-10</td> <td>1073741823</td> <td>N'</td> </tr> <tr> <td>nvarchar</td> <td>-9</td> <td>4000</td> <td>N'</td> </tr> <tr> <td>sysname</td> <td>-9</td> <td>128</td> <td>N'</td> </tr> <tr> <td>date</td> <td>-9</td> <td>10</td> <td>'</td> </tr> </tbody> </table>	TYPE_NAME	DATA_TYPE	PRECISION	LITERAL	sql_variant	-150	8000	NULL	uniqueidentifier	-11	36	'	ntext	-10	1073741823	N'	xml	-10	1073741823	N'	nvarchar	-9	4000	N'	sysname	-9	128	N'	date	-9	10	'	
TYPE_NAME	DATA_TYPE	PRECISION	LITERAL																																
sql_variant	-150	8000	NULL																																
uniqueidentifier	-11	36	'																																
ntext	-10	1073741823	N'																																
xml	-10	1073741823	N'																																
nvarchar	-9	4000	N'																																
sysname	-9	128	N'																																
date	-9	10	'																																
sp_depends	Shows where a particular table is used in your database (for example, sp_depends 'Film') or which tables and columns a procedure references (eg sp_depends 'spExample').	<table border="1"> <thead> <tr> <th>name</th> <th>type</th> </tr> </thead> <tbody> <tr> <td>dbo.spExample</td> <td>stored procedure</td> </tr> <tr> <td>dbo.spNewName</td> <td>stored procedure</td> </tr> <tr> <td>dbo.View_1</td> <td>view</td> </tr> </tbody> </table>	name	type	dbo.spExample	stored procedure	dbo.spNewName	stored procedure	dbo.View_1	view																									
name	type																																		
dbo.spExample	stored procedure																																		
dbo.spNewName	stored procedure																																		
dbo.View_1	view																																		



You can see more examples of the above at this blog:

http://www.wiseowl.co.uk/blog/s2522/system_stored_procedures.htm

2.7 Getting Help on SQL

Although every programmer will have their own way to get help, here are couple of general tips.

Context-Sensitive Help

You can press **F1** on any keyword (or collection of keywords) to show help in your web browser:

Select a word or a selection of words and then press **F1** ...

... to get SSMS to suggest (in this case, very appropriate) help.

CREATE PROCEDURE (Transact-SQL)

Updated: December 16, 2016

THIS TOPIC APPLIES TO: SQL Server (starting with 2008) Azure SQL Database Azure Warehouse

Creates a Transact-SQL or common language runtime (CLR) stored procedure in SQL Server. Stored programming languages in that they can:

Tips on Googling





If you're reading this, you probably don't need much help on using search engines. Here's our advice for how to get help on any SQL topic:

Typing **T-SQL** (short for *Transact-SQL*) ensures you'll get help only on SQL as used within Management Studio, and not on the MySql or Oracle SQL variants.

Whatever it is that you want help on (in this case how to use the **CHARINDEX** function in SQL).

It's sometimes worth adding this to omit Microsoft sites, which tend to be more technical reference than user guide (and in any case you could have gone to the Microsoft help site just by pressing **F1** on a word, as above).

WHAT WE DO

	 ONLINE TRAINING	 MANCHESTER OR LONDON	 AT YOUR OFFICE	 BESPOKE CONSULTANCY	
OFFICE 365	Microsoft Excel	✓	✓	✓	✓
	VBA macros	✓	✓	✓	✓
	Office Scripts	✓		✓	
	Microsoft Access				✓
POWER PLATFORM	Power BI and DAX	✓	✓	✓	✓
	Power Apps	✓		✓	
	Power Automate	✓	✓	✓	✓
SQL SERVER	Reporting Services	✓	✓	✓	✓
	Report Builder	✓		✓	✓
	Integration Services	✓	✓	✓	✓
	Analysis Services	✓		✓	
CODING LANGUAGES	SQL	✓	✓	✓	✓
	Visual C#	✓	✓	✓	✓
	Python	✓	✓	✓	✓



WiseOwl
Training

