CHAPTER 6

Automating Deployment with SQLCMD

So far, we've created nine scripts: one to create the database, and eight more containing tables and foreign key relationships. You don't want to ask your DBAs to load and execute each script individually. This is where the SQLCMD tool comes in—you can create a simple script that will allow users to execute all of your scripts with a simple press of F5.

You can also use SQLCMD to roll back your changes if necessary, so once you've created the apply SQLCMD script, we'll look at creating some rollback scripts.

What Is SQLCMD?

You were very briefly introduced to SQLCMD way back in Chapter 1. It's a tool, supplied as part of SQL Server, that allows you to manage various aspects of your server. You can also execute T-SQL commands via this tool. SQLCMD can be used in two ways: via the command line, or via SQLCMD Mode in SSMS. We'll use the latter.

We want to use SQLCMD in an extremely simple manner; our intention is to do nothing more than execute multiple scripts in one go. This is the absolute minimum of what SQLCMD is capable of and it's well worth investigating further. Visit `https://msdn.microsoft.com/en-us/library/ms162773.aspx` for more details on what SQLCMD can do.

Why We Need a SQLCMD Script

Assume we need to make a change to script `02 - Create Contacts Table.sql`. We want to rename a column. Open `c:\temp\sqlbasics\02 - Create Contacts Table.sql` and run it. You'll see a couple of error messages:

```
Msg 3726, Level 16, State 1, Line 5
Could not drop object 'Contacts' because it is referenced by a FOREIGN KEY constraint.
Msg 2714, Level 16, State 6, Line 8
There is already an object named 'Contacts' in the database.
```

The first error happens because the `Contacts` table cannot be dropped. It cannot be dropped because the `Contacts` table is linked to other tables via foreign keys. The other tables must be dropped before the `Contacts` table can be dropped.

The second error happens because the `CREATE TABLE` statement has tried to execute. It failed because the table already exists, as the table wasn't dropped in the first place.

Close the `02 - Create Contacts Table.sql` script, and open a New Query Window.

Rollback Scripts

We've already seen that we cannot run the `Contacts` Table script, and it's fair to assume there will be other scripts in our apply folder that will not work at the moment. So before we create our SQLCMD scripts we'll create our rollback scripts.

Open Windows Explorer and navigate to `c:\temp\sqlbasics\rollback`. You should have one script in there, `01 - Create AddressBook Database Rollback.sql`. We created this in Chapter 3 to roll back the database creation. It looks like this:

```
USE Master;

IF EXISTS (SELECT 1 FROM sys.databases WHERE [name] = 'AddressBook')
BEGIN
DROP DATABASE AddressBook;
END
```

```
GO
```

It does nothing more than check if the **AddressBook** database exists. If it does, the database is dropped.

We need to create eight more rollback scripts, one for each of our table scripts. We'll start from script `02`. In the New Query Window you opened, type the rollback script:

```
USE AddressBook;

IF EXISTS (SELECT 1 FROM sys.tables WHERE [Name] = 'Contacts')
BEGIN
DROP TABLE dbo.Contacts;
END;

GO
```

This is pretty much the same as the apply script, except it doesn't have the `CREATE TABLE` statement in it. Save this script as `c:\temp\sqlbasics\rollback\02 - Create Contacts Table Rollback.sql`. Then, for all the other scripts, follow these steps:

- Open New Query Window
- Enter the script:

```
USE AddressBook;

IF EXISTS (SELECT 1 FROM sys.tables WHERE [Name] = 'ContactNotes')
BEGIN
DROP TABLE dbo.ContactNotes;
END;

GO
```

- Save the script as `c:\temp\sqlbasics\rollback\03 - Create ContactNotes Table Rollback.sql`
- Open New Query Window
- Enter the script:

```
USE AddressBook;

IF EXISTS (SELECT 1 FROM sys.tables WHERE [Name] = 'Roles')
BEGIN
DROP TABLE dbo.Roles;
END;

GO
```

- Save the script as `c:\temp\sqlbasics\rollback\04 - Create Roles Table Rollback.sql`
- Open New Query Window
- Enter the script:

```
USE AddressBook;

IF EXISTS (SELECT 1 FROM sys.tables WHERE [Name] = 'ContactRoles')
BEGIN
DROP TABLE dbo.ContactRoles;
END;

GO
```

- Save the script as `c:\temp\sqlbasics\rollback\05 - Create ContactRoles Table Rollback.sql`
- Open New Query Window
- Enter the script:

```
USE AddressBook;

IF EXISTS (SELECT 1 FROM sys.tables WHERE [Name] = 'ContactAddresses')
BEGIN
DROP TABLE dbo.ContactAddresses;
END;

GO
```

- Save the script as `c:\temp\sqlbasics\rollback\06 - Create ContactAddresses Table Rollback.sql`
- Open New Query Window
- Enter the script:

```
USE AddressBook;

IF EXISTS (SELECT 1 FROM sys.tables WHERE [Name] = 'PhoneNumberTypes')
BEGIN
DROP TABLE dbo.PhoneNumberTypes;
END;

GO
```

- Save the script as `c:\temp\sqlbasics\rollback\07 - Create PhoneNumberTypes Table Rollback.sql`
- Open New Query Window
- Enter the script:

```
USE AddressBook;

IF EXISTS (SELECT 1 FROM sys.tables WHERE [Name] = 'ContactPhoneNumbers')
BEGIN
DROP TABLE dbo.ContactPhoneNumbers;
END;

GO
```

- Save the script as `c:\temp\sqlbasics\rollback\08 - Create ContactPhoneNumbers Table Rollback.sql`
- Open New Query Window
- Enter the script:

```
USE AddressBook;

IF EXISTS (SELECT 1 FROM sys.tables WHERE [Name] = 'ContactVerificationDetails')
BEGIN
DROP TABLE dbo.ContactVerificationDetails;
END;

GO
```

- Save the script as `c:\temp\sqlbasics\rollback\09 - Create ContactVerificationDetails Table Rollback.sql`

That was a fair bit of typing. If you look at your rollback folder in Windows Explorer you should see nine files, which are shown in Figure 6-1.

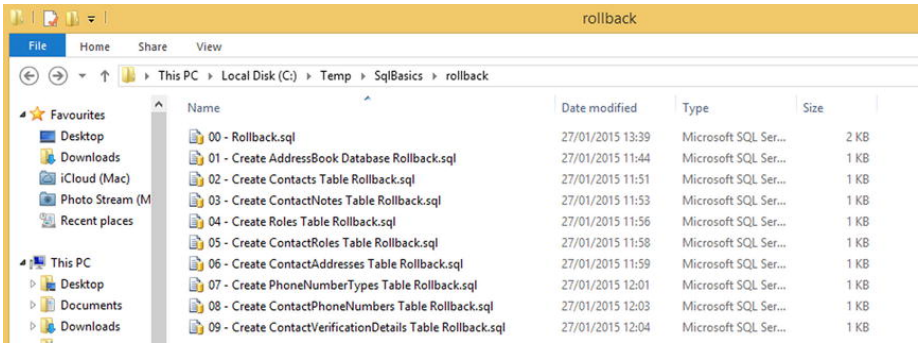

Figure 6-1. *Rollback files*

We have nine apply files and nine rollback files. Now we can look at creating a SQLCMD file for rollback purposes.

Creating the SQLCMD Rollback Script

In SSMS, press Ctrl+N to open a New Query Window. Go to the **Query** menu and click the **SQLCMD Mode** option you can see in Figure 6-2.

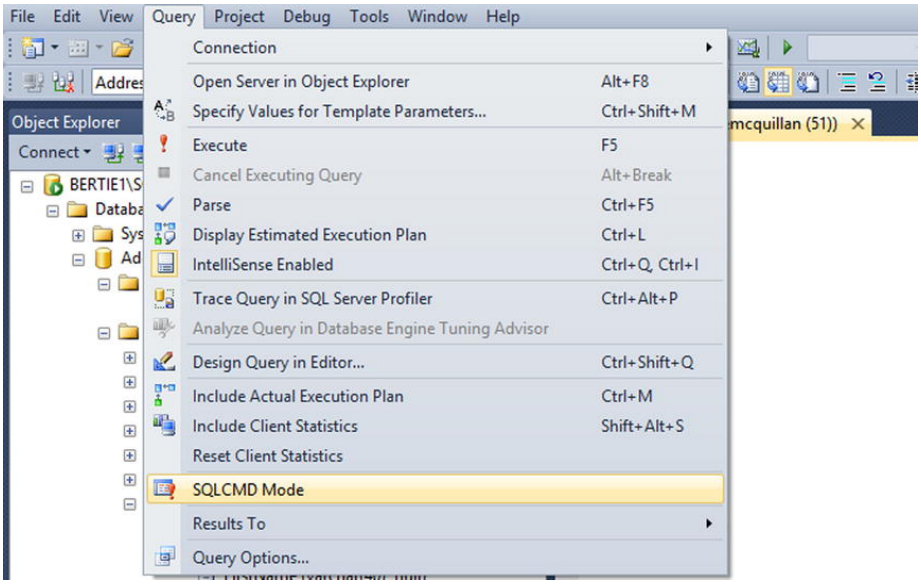

Figure 6-2. *Turning on SQLCMD mode in SSMS*

Absolutely nothing happens when you click SQLCMD mode. Don't worry about that.

Type this code:

```
USE master;

:setvar path "C:\temp\sqlbasics\rollback\"

:setvar currentFile "09 - Create ContactVerificationDetails Table Rollback.sql"
PRINT 'Executing $(path)$(currentFile)';
:r $(path)$(currentFile)
```

The `USE` statement we've seen before, but everything else is brand new. The lines beginning with a colon are bold (they will be highlighted gray on screen) and do not end with a semicolon. This shows they are SQLCMD commands. If you turn SQLCMD mode off the highlighting will disappear and red underlines will appear instead, showing the code is not valid T-SQL.

The first line begins with `:setvar`. This declares a variable, called `path`, and we assign the value `c:\temp\sqlbasics\rollback\` to it. This holds the path in which our rollback files reside. The end `\` is

very important, as we'll concatenate the file name to the end of the path when we execute the script in a moment.

The next three lines are grouped together, indicating they all act together to perform an action. We start with another `:setvar`, this time setting the name of the file we want to process—in this case, script `09` to roll back the creation of the `ContactVerificationDetails` table. The following line starts with `:r`. This is a little more difficult to figure out by looking at it; `:setvar` can be seen as a shortened version of *set variable*. What could `:r` mean?

Next, we have a `PRINT` statement. This just outputs the contents of the variables. It is here to tell us what the `:r` statement is going to do.

Finally, we have the `:r` command. `:r` appends the contents of the specified file to the statement cache. In other words, SQL Server will execute the contents of the file provided. This tells us what `:r` does, but it gives us no indication of why the command is called `:r`. My best guess is *r* stands for "run" (one other developer I know says it means "return"). I've never been able to confirm this, so if anybody has any other ideas feel free to e-mail me!

The `:r` command we are using—`:r $(path)$(currentFile)`—will obtain the path to the SQL file by joining the contents of the `path` and `currentFile` variables. Note that we had to wrap the variable names in brackets, preceded by a `$` sign. This is how SQLCMD parses variables.

Run this by pressing F5. You should see:

```
Executed C:\temp\SqlBasics\rollback\09 - Create ContactVerificationDetails Table Rollback.sql
```

If you refresh your Tables node, you'll see the `ContactVerificationDetails` table has disappeared. We'll complete this script to roll the entire database back. Add the following under the code you've already entered.

```
USE master;

:setvar path "C:\temp\SqlBasics\rollback\"

:setvar currentFile "09 - Create ContactVerificationDetails Table Rollback.sql"
PRINT 'Executing $(path)$(currentFile)';
:r $(path)$(currentFile)

:setvar currentFile "08 - Create ContactPhoneNumbers Table Rollback.sql"
PRINT 'Executing $(path)$(currentFile)';
:r $(path)$(currentFile)

:setvar currentFile "07 - Create PhoneNumberTypes Table Rollback.sql"
PRINT 'Executing $(path)$(currentFile)';
:r $(path)$(currentFile)

:setvar currentFile "06 - Create ContactAddresses Table Rollback.sql"
PRINT 'Executing $(path)$(currentFile)';
:r $(path)$(currentFile)

:setvar currentFile "05 - Create ContactRoles Table Rollback.sql"
PRINT 'Executing $(path)$(currentFile)';
:r $(path)$(currentFile)

:setvar currentFile "04 - Create Roles Table Rollback.sql"
PRINT 'Executing $(path)$(currentFile)';
:r $(path)$(currentFile)

:setvar currentFile "03 - Create ContactNotes Table Rollback.sql"
PRINT 'Executing $(path)$(currentFile)';
:r $(path)$(currentFile)

:setvar currentFile "02 - Create Contacts Table Rollback.sql"
PRINT 'Executing $(path)$(currentFile)';
:r $(path)$(currentFile)

:setvar currentFile "01 - Create AddressBook Database Rollback.sql"
PRINT 'Executing $(path)$(currentFile)';
:r $(path)$(currentFile)

PRINT 'All rollback scripts successfully executed.';

USE master;
```

This is the same code we've already seen, just repeated for each file we want to process. Note we're processing the files in reverse order, `09` to `01`. Remember this is a rollback, so we need to remove each database object in the order we added it. When we apply, we'll apply from `01` to `09`, so we need to roll back from `09` to `01`.

THE IMPORTANCE OF ORDER

Numbering your scripts is completely optional, but it is a really good idea and will make managing script execution much easier. By numbering, you can clearly see the order in which scripts should be executed. Likewise, you can simply reverse the numeric order to create a rollback script.

We finish up with a `USE` master statement. This gives the focus back to the master database at the end of the script.

Save this script to `c:\temp\sqlbasics\rollback\00 - Rollback.sql`. When you feel brave enough, run it—this will delete the entire **AddressBook** database. Figure 6-3 shows a successful execution of the rollback script.

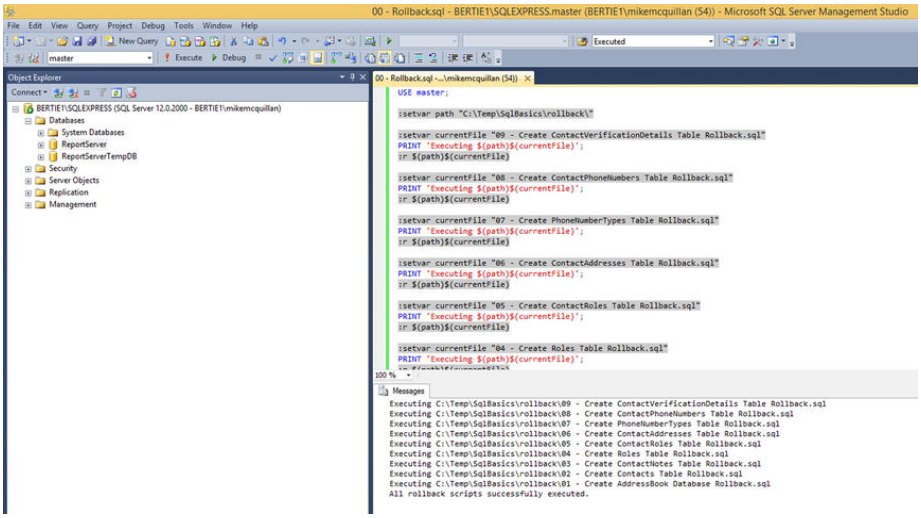

Figure 6-3. *Executing a rollback script in SQLCMD mode*

On to the apply version!

Creating the SQLCMD Apply Script

This is pretty much exactly the same script, except we point it at the apply path and execute the scripts in ascending numerical order. It should be obvious now why we've been numbering the scripts as we go: it makes it easy to identify the order in which they should be executed.

Here's the apply script; type this into a New Query Window. Don't forget to turn on SQLCMD mode.

```
USE master;

:setvar path "C:\temp\SqlBasics\apply\"

:setvar currentFile "01 - Create AddressBook Database.sql"
PRINT 'Executing $(path)$(currentFile)';
:r $(path)$(currentFile)

:setvar currentFile "02 - Create Contacts Table.sql"
PRINT 'Executing $(path)$(currentFile)';
:r $(path)$(currentFile)

:setvar currentFile "03 - Create ContactNotes Table.sql"
PRINT 'Executing $(path)$(currentFile)';
:r $(path)$(currentFile)

PRINT 'Executing $(path)$(currentFile)';
:setvar currentFile "04 - Create Roles Table.sql"
```

```
:r $(path)$(currentFile)

:setvar currentFile "05 - Create ContactRoles Table.sql"
PRINT 'Executing $(path)$(currentFile)';
:r $(path)$(currentFile)

:setvar currentFile "06 - Create ContactAddresses Table.sql"
PRINT 'Executing $(path)$(currentFile)';
:r $(path)$(currentFile)

:setvar currentFile "07 - Create PhoneNumberTypes Table.sql"
PRINT 'Executing $(path)$(currentFile)';
:r $(path)$(currentFile)

:setvar currentFile "08 - Create ContactPhoneNumbers Table.sql"
PRINT 'Executing $(path)$(currentFile)';
:r $(path)$(currentFile)

:setvar currentFile "09 - Create ContactVerificationDetails Table.sql"
PRINT 'Executing $(path)$(currentFile)';
:r $(path)$(currentFile)

PRINT 'All apply scripts successfully executed.';

USE master;
```

Save this as `c:\temp\sqlbasics\apply\00 - Apply.sql`. Run it and, as in Figure 6-4, your database will magically be restored!

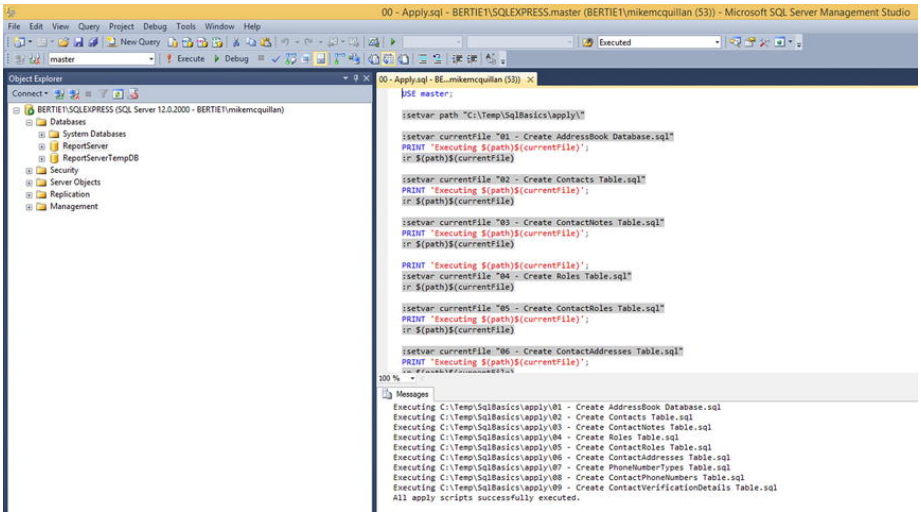

Figure 6-4. *Executing an apply script in SQLCMD mode*

You can run the apply and rollback scripts as many times as you want. Run the rollback to drop the database, then run the apply to restore it. Repeat this as many times as you like!

Summary

Thanks to SQLCMD, you now have the ability to create and drop our database with ease. SQLCMD can do much more than you've seen here and is a tool that you should learn to love. Read up on the MSDN documentation at `https://msdn.microsoft.com/en-us/library/ms162773.aspx` and turn yourself into a SQLCMD expert!

We will now finish off our tables, by ensuring only good data can be entered into the tables. We'll use a combination of `NULL` and constraints to do this.

PREV
Chapter 5 : Putting Good Tables…

NEXT
Chapter 7 : NULLs and Table Con..

R

Sign Out