

# Static Data



## Data population strategies

If you're interested in the various techniques for seeding your database in general, you may like to start by reading [this article on data population strategies](#).

One thing to consider when source-controlling your database is how to maintain static/lookup data within your project.

This can include data that is needed by the calling application to work such as zip/postal codes, country lists or even application settings.

ReadyRoll offers two distinct ways of handling static data:

- **Online method** – use the SQL Data Compare engine (built into ReadyRoll) to generate migrations containing INSERT/UPDATE/DELETE statements  
This is the simplest way to edit static data. Simply flag one or more tables for data tracking, and you can edit the data directly in the target database and then synchronize with your database project to generate a migration script.
- **Offline method** – use an open-source tool to generate re-usable MERGE statements  
This method has a higher learning curve than the online method but offers more flexibility. The advantage of working offline is that you can store the data in a single file, rather than generating a new file with each change, allowing you to branch and merge the data in source control.



You can choose to use either the online or offline editing approaches above, or a combination of both, in the one database project.

For example, you may use the online method on tables that contain consistent data between all environments, and use the offline method on tables where the data changes between environments.



## Handling larger data sets

If you're looking to seed a table with a larger amount of data (10,000+ rows) then you may find that the approaches outlined in this article unsuitable, as INSERT-ing many rows with literal values does not tend to scale very well. For larger data sets, we recommend including a seed file in your project and using the BULK INSERT statement to upload the data. [Read more about using the BULK INSERT method](#).

## Editing Static Data Online

In addition to synchronizing schema and code objects, the ReadyRoll tool-window in Visual Studio can also synchronize static data..

This allows you to edit table data in an online manner, e.g. with SQL Management Studio, and then import changes back into your database project to generate INSERT/UPDATE/DELETE statements within a new migration.

To mark a table for data tracking:

1. Open your Database Project in Visual Studio, and switch to the ReadyRoll tool-window (shown below).
2. Click **View Pending Changes** to display the list of tables available for data tracking. Expand the *Identical Objects* group to locate existing tables.
3. Right-click a table from the list and check **Include Table data**.  
*A notification bar should appear, indicating that a Refresh is pending.*
4. Refresh the pending change list. This time, ReadyRoll perform a comparison that will include data from the specified table(s).



## Primary Key requirement

In order to track the data within your tables, the table must include a primary key. If a primary key is not present in a given table, the data within the table will be ignored (not imported).

You can now preview and import data from the specified tables.

The screenshot shows a SQL Server Enterprise Manager interface. The top pane displays a SQL script for a MERGE operation on the [Purchasing].[ShipMethod] table. The script includes a SET IDENTITY\_INSERT statement, a MERGE INTO statement with a VALUES clause containing six rows of data, and an UPDATE SET clause for the WHEN MATCHED THEN condition. The bottom pane shows a comparison between the local AdventureWorks database and a remote server. A table named ShipMethod is highlighted, and a context menu is open, showing options like 'View Object Differences', 'Include Table Data', 'Revert Selected Object(s)...', and 'View Revert Script...'. The table comparison window also shows that there are changes pending import from the remote server.



### Working with larger data sets

When choosing tables to track for data changes, it is important to bear in mind that ReadyRoll is optimized for working with smaller sets of static data (< 1MB table size). This is to ensure that your database projects always build quickly, as larger datasets can produce a significant amount of T-SQL code which can cause performance issues during project build/deployment.

A warning will display within the *ReadyRoll* tool-window if data 'marked' table exceeds 1MB.

For larger data sets, we recommend using the [data seed method](#) with **BULK INSERT** method instead.



### Limitations in Continuous Integration / Deployment

Please note that ReadyRoll's static data tracking is not currently supported within the [drift correction feature](#), so the drift report will not report on any data differences. Likewise, the [deployment preview feature](#) which shows object-level changes that are pending deployment (e.g. in Octopus / VSTS / TFS), does not yet support the previewing of data changes.

## Editing Static Data Offline

Editing offline means that you start by preparing a data change script before actually deploying the changes to the database.

In this example, we'll use the [MERGE statement](#) to manage the deployment of our static data. What makes MERGE so useful is not only its ability to insert, update or delete data in one succinct, atomic operation but also the fact that the statements are re-runnable. This allows for the source script to be edited and re-used for deployment to all target environments. It also gives you the ability to include [SQLCMD variables](#) in place of literal values, allowing you to centralize the deployment of configuration data ([read more about deploying environment-specific data](#)).

To generate MERGE statements, we will use the **sp\_generate\_merge** stored procedure utility. The script that is generated from the utility can be pasted into a *Post-Deployment* script in your ReadyRoll project, as explained below.



### Limitations of the offline method

There are a few draw-backs of the offline method to consider prior to selecting this approach:

- Non-determinism of the `MERGE` statement: before actually running the deployment against your target environment, it can be difficult to know what changes will be applied (if any). Worst case scenario, you could hit one of the [documented issues in MERGE](#)
- The workflow isn't necessarily the most natural way to edit data, as it requires running the utility proc and copying+pasting the output back into the original file. Editing the file directly is an alternative, but isn't the most user-friendly experience especially with large amounts of reference data
- Coordinating changes to both the schema and data within the reference table can be quite difficult, given that your schema changes will be performed in a separate part of your deployment (i.e. [multi-use scripts](#) run only after all pending [migrations](#) have been executed). For example, if you need to update an ID value in the lookup table prior to adding a foreign key to the schema, then that update would need to be hand-coded and included in your project separately to your `MERGE` statement.

If you foresee that these limitations may be problematic for your deployments, then it may be worth considering the online approach to source controlling your static data instead.

## Installing the "sp\_generate\_merge" proc

Download the ["master.dbo.sp\\_generate\\_merge" stored procedure](#) and install it by simply running the script on your dev SQL Server instance. This will install the utility as a system procedure within the [master] database so that it can be used within all of your user databases.



### Open Source Project

Note that you do not need to add this script to your ReadyRoll database project, nor do you need to deploy the stored procedure to any server other than your Development environment.

Full details of the `"sp_generate_merge"` stored procedure [can be found on GitHub](#).

## Generating a MERGE statement from existing data

In this step we'll generate a single `MERGE` statement containing all the records from the "Region" table in the Northwind database:

The screenshot shows a SQL query window titled "SQLQuery6.sql - W...aniel Nolan (55)". The query is:

```
SELECT [RegionID]
, [RegionDescription]
FROM [Northwind].[dbo].[Region]
```

Below the query window is a "Results" tab showing a grid of data with two columns: "RegionID" and "RegionDescription".

	RegionID	RegionDescription
1	1	Eastern
2	2	Western
3	3	Northern
4	4	Southern

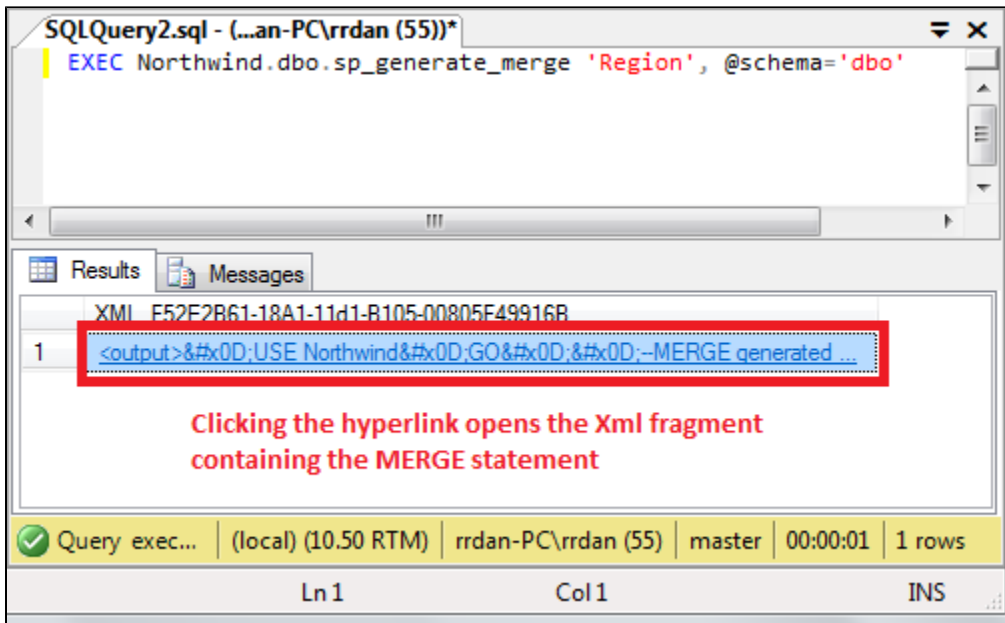
At the bottom of the window, a status bar shows "50 RTM | WIN-HVNLUT6MRH\Daniel... master 00:00:00 4 rows".

Firstly, open SQL Management Studio and connect to your development database server and ensure that your SQL client is configured to send results to grid, rather than text.

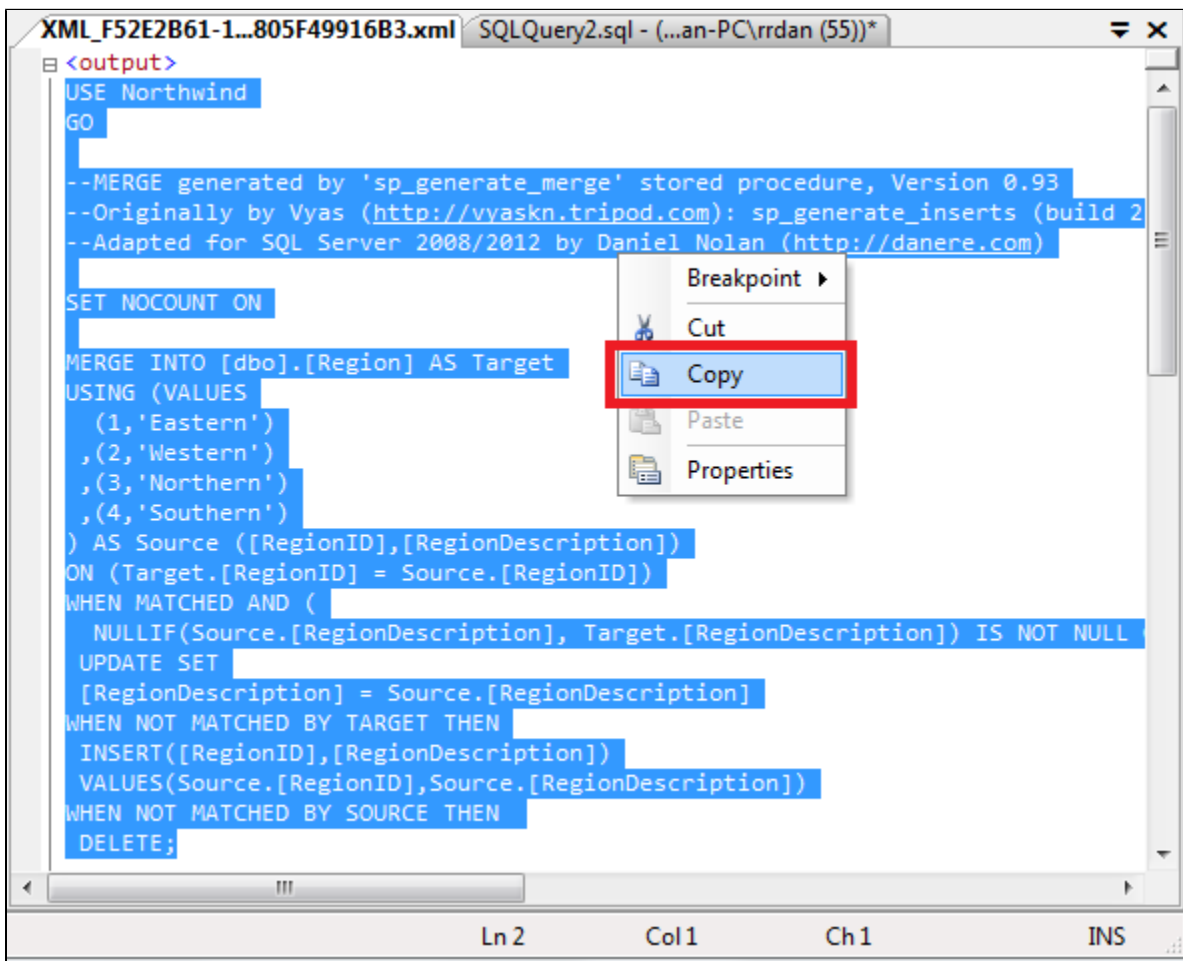
Execute the stored procedure, providing the source table name as a parameter. If your table is in a non-default schema, be sure to supply the `@schema` parameter with its name.

**For example:**

```
EXEC Northwind.dbo.sp_generate_merge 'Region', @schema='dbo'
```



Click the hyperlink in the result set to open up the Xml fragment in a new document window.

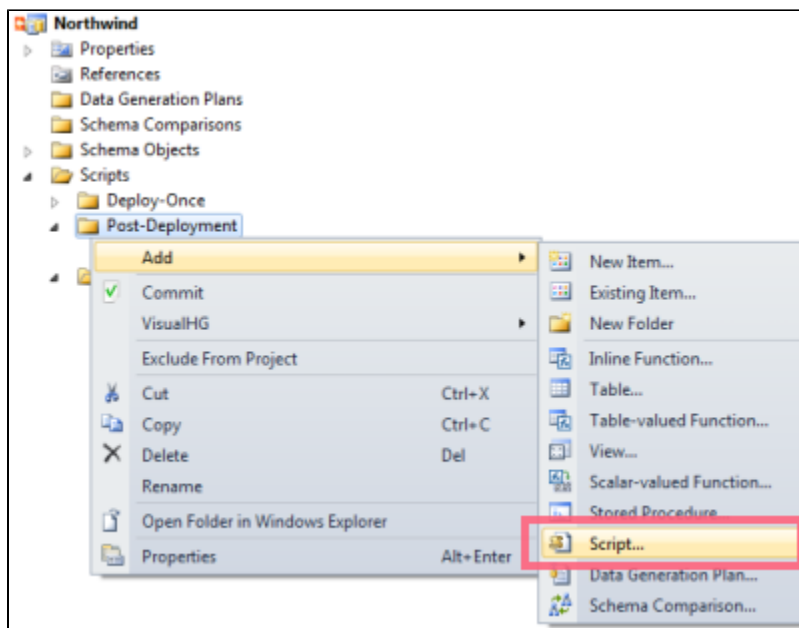


Copy the document contents (excluding the Xml tags) to the clipboard and switch back to Visual Studio.

In the next step we'll create a script to be executed during each deployment which will ensure that the table is always in-sync.

## Add a deployment script to your project

Within Visual Studio, open the Solution Explorer tool-window (*View... Solution Explorer*) and expand your ReadyRoll database project's **Post-Deployment** sub-folder.



Right-click the **Post-Deployment** sub-folder and select *Add... Script*. In the New Script dialog, select the *Script (Not In Build)* template and specify a name that includes the next sequential number, eg. *01\_Populate\_Region\_Data.sql*.

#### Script naming

Naming your Pre/Post-Deployment scripts in this way isn't mandatory for your project to build, however we recommend sequentially numbering your scripts to make it easier to explicitly define the order of execution. This is particularly important if there are interdependencies between your static data tables.

Paste the SQL copied in the previous step into the newly created script file.

With the benefit of syntax highlighting, we can now take a bit more of a closer look at what is going on in the generated script:

```
--MERGE generated by 'sp_generate_merge' stored procedure, Version 0.9
--Originally by Vyas (http://vyaskn.tripod.com): sp_generate_inserts (build 22)
--Adapted for SQL Server 2008 by Daniel Nolan (http://danere.com)

SET NOCOUNT ON      Reporting of rows affected is switched off, as this is handled after the MERGE

MERGE INTO [Region] AS Target ← Table to be populated
USING (VALUES

    (1, 'Eastern')
    , (2, 'Western')
    , (3, 'Northern')
    , (4, 'Southern')

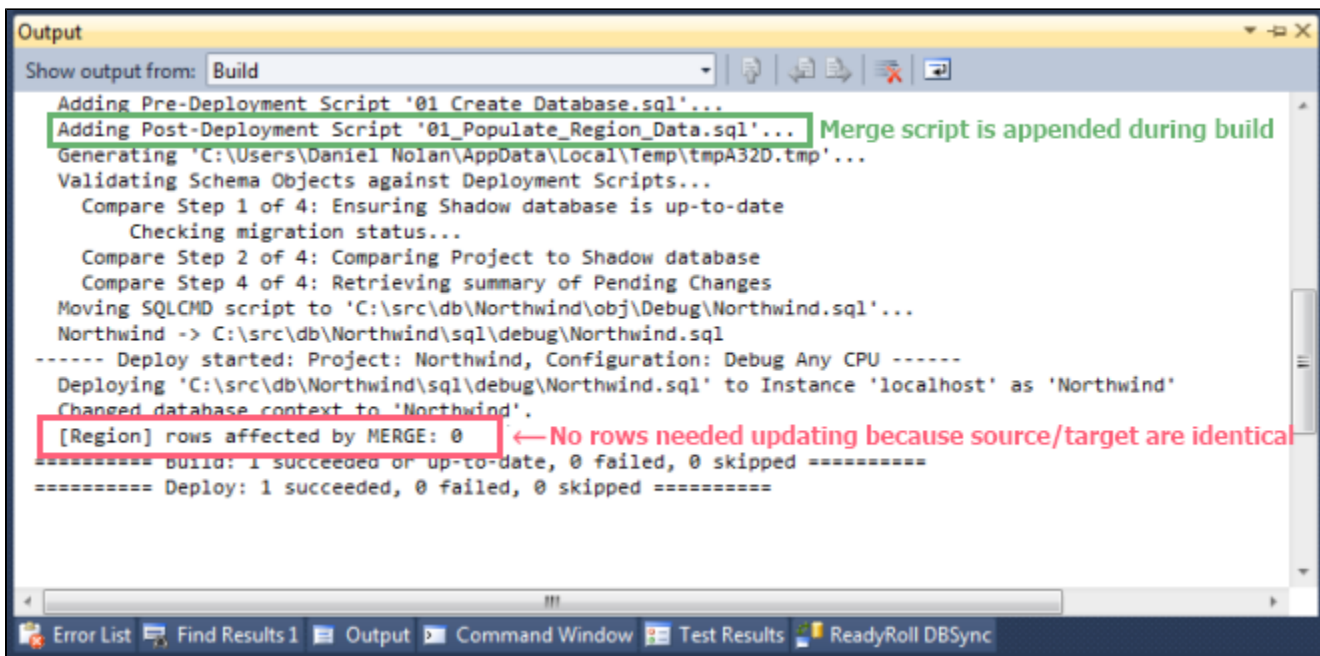
) AS Source ([RegionID],[RegionDescription]) ← List of column names to be associated with the above dataset
ON (Target.[RegionID] = Source.[RegionID]) ← Join Source & Target on Primary Key so that existing rows can be matched
WHEN MATCHED AND (Target.[RegionDescription] <> Source.[RegionDescription]) THEN
    UPDATE SET
        [RegionDescription] = Source.[RegionDescription] ← Only UPDATE if data differs between Source data & Target table
WHEN NOT MATCHED BY TARGET THEN
    INSERT([RegionID],[RegionDescription])
    VALUES(Source.[RegionID],Source.[RegionDescription])
WHEN NOT MATCHED BY SOURCE THEN
    DELETE;

GO

DECLARE @mergeError int
        , @mergeCount int
SELECT @mergeError = @@ERROR, @mergeCount = @@ROWCOUNT
IF @mergeError != 0
    BEGIN
        PRINT 'ERROR OCCURRED IN MERGE FOR [Region]. Rows affected: ' + CAST(@mergeCount AS VARCHAR(100));
    END
ELSE
    BEGIN
        PRINT '[Region] rows affected by MERGE: ' + CAST(@mergeCount AS VARCHAR(100));
    END
GO

SET NOCOUNT OFF
GO
```

To test your new script, deploy your database project to your development SQL Server (*Build... Deploy Solution*). The Output window should display results similar to below:



```
Output
Show output from: Build
Adding Pre-Deployment Script '01 Create Database.sql'...
Adding Post-Deployment Script '01_Populate_Region_Data.sql'... Merge script is appended during build
Generating 'C:\Users\Daniel Nolan\AppData\Local\Temp\tmpA32D.tmp'...
Validating Schema Objects against Deployment Scripts...
  Compare Step 1 of 4: Ensuring Shadow database is up-to-date
    Checking migration status...
  Compare Step 2 of 4: Comparing Project to Shadow database
  Compare Step 4 of 4: Retrieving summary of Pending Changes
Moving SQLCMD script to 'C:\src\db\Northwind\obj\Debug\Northwind.sql'...
Northwind -> C:\src\db\Northwind\sql\debug\Northwind.sql
----- Deploy started: Project: Northwind, Configuration: Debug Any CPU -----
Deploying 'C:\src\db\Northwind\sql\debug\Northwind.sql' to Instance 'localhost' as 'Northwind'
Changed database context to 'Northwind'.
[Region] rows affected by MERGE: 0 ← No rows needed updating because source/target are identical
===== Build: 1 succeeded or up-to-date, 0 failed, 0 skipped =====
===== Deploy: 1 succeeded, 0 failed, 0 skipped =====
```



#### Making Changes to Static Data

To see how the script can be re-used to repeatedly deploy changes in your static data, try making changes to the data in the script and re-run the deployment. The rows affected by the MERGE should reflect your additions and modifications to the file.

If you prefer to initiate changes to data by editing the table data directly in SSMS, you can do so by simply re-running the *sp\_generate\_merge* procedure after making changes to the live data. Copy+Paste the generated code as you did in the previous steps to update your Post-Deployment script with the new changes.

## Using MERGE to deploy configuration data

Using the offline method outlined above gives you the ability to include [SQLCMD variables](#) in place of literal values, allowing data from the deployment system to be passed in and stored within your tables (e.g. environment-specific variables from Octopus).

This means you could use your deployment system to centralize the storage of configuration settings, giving you a systematic and repeatable process for propagating updates to configuration data.

[Read more about deploying config data in this tutorial.](#)