

Continuous integration for databases using Red Gate SQL tools

Contents

Introduction.....	3
Solving the source control problem	5
Automation with MSBuild, NAnt, and Powershell.....	11
Conclusion.....	15

Introduction

This article examines the challenge of integrating databases with the development cycle, and describes how a combination of Red Gate tools and tools like MSBuild, NAnt, and Powershell can be used to automate the process.

The first section provides simple examples using the command line interfaces of Red Gate SQL Compare and SQL Data Compare. The second section looks at some more complex automation techniques.

This introduction addresses the following questions:

- What is continuous integration? (page 3)
- How are databases different? (page 4)
- How can Red Gate tools help? (page 4)

What is continuous integration?

Continuous integration is the process of ensuring that all code and related resources in a development project are integrated regularly and tested by an automated build system. Code changes are checked into source control, any conflicts are resolved, and significant changes trigger an automated build with unit tests and rapid feedback. A stable current build is consistently available, and if a build fails, it is fixed rapidly and re-tested.

A continuous integration server uses a build script to execute a series of commands that build an application. Generally, these commands clean directories, run a compiler on source code, and execute unit tests. However, build scripts can be extended to perform additional tasks, such as deploying the application or updating a database with SQL Compare.

Continuous integration is now an established development practice. Its benefits were outlined by Martin Fowler in the article [Continuous Integration](#)

...there is a stable piece of software that works properly and contains few bugs. Everybody develops off that shared stable base and never gets so far away from that base that it takes very long to integrate back with it. Less time is spent trying to find bugs because they show up quickly.

[...]

The basic rule of thumb is that you should be able to walk up to the project with a virgin machine, do a checkout, and be able to fully build the system. Only a minimal amount of things should be on the virgin machine - usually things that are large, complicated to install, and stable.

For many software projects, this will include a database. Fowler recommends that "getting the database schema out of the repository and firing it up in the execution environment" should be part of the automatic build process. However, this is not always simple.

How are databases different?

The principal difficulty of continuous integration for databases is the lack of a simple way to keep a database in source control and deploy it to a target server.

Because DML and DDL queries modify the current state of a database, there is no source code to compile. Migration and deployment therefore rely on creating scripts specifically for that purpose.

The lack of source code makes it difficult to maintain a current stable version in source control. Creation and migration scripts can be checked into the source control repository, but the actual creation of these scripts is not considered to be part of the database development cycle.

Migration scripts may contain ALTER and UPDATE statements to synchronize the target version of the database with the development version; alternatively, the scripts may create a new database. Where changes are deployed to an existing database, all differences and dependencies must be accounted for. In some production deployments, this involves multiple targets with different schemas and data. In either case, the manual process is time consuming and prone to errors.

Object creation scripts can be generated relatively simply (for example using Microsoft SQL Server Management Studio), but referential integrity is difficult to maintain. Objects must be created (and populated with data) in the correct order, otherwise they may be invalid. As dependency chains can be complex, third party tools are often required. Data migration or test data creation, similarly, are tedious and time consuming operations when performed manually.

How can Red Gate tools help?

Red Gate offers the following tools for automating database development:

- SQL Compare
Compares and synchronizes database schema.
- SQL Data Compare
Compares and synchronizes database data.
- SQL Data Generator
Generates realistic test data based on your database schema.

The professional editions of SQL Compare and SQL Data Compare enable you to create, compare, and synchronize folders of SQL scripts representing a database's schema and data. The process is rapid, accurate, and - crucially - can be automated using the command line interface.

A representation of the database can therefore be created and automatically maintained in a source control repository without manual intervention.

Solving the source control problem

Using the command line interface of Red Gate's SQL comparison tools, it is easy to reduce scripting operations to a single click or a scheduled task.

There are three stages to keeping your database in source control with Red Gate tools:

1. Setting up source control (page 6)

Create a scripts folder and check it into the repository.

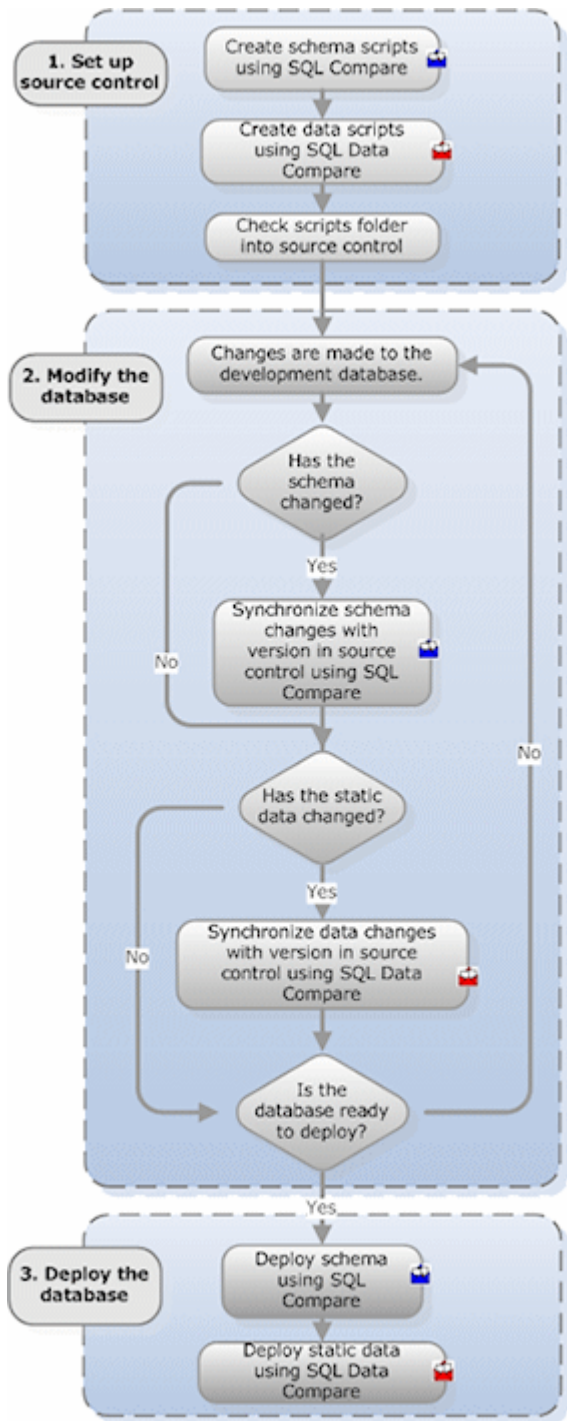
2. Synchronizing development changes (page 7)

Periodically, changes to the development database are compared and synchronized with the scripts in source control, maintaining a stable current version of the database. Development continues, and the scripts are updated automatically, without manual intervention.

3. Deploying to the test environment (page 9)

When a build is triggered, the scripts in source control are deployed to the target database using SQL Compare and SQL Data Compare.

The diagram below gives an overview of the whole process:



Example: setting up source control

In this example, changes are made to the schema and data of the database WidgetDev. These changes are synchronized with the database in source control, and eventually deployed to the testing database WidgetTest.

WidgetDev is maintained in source control as the scripts folder WidgetDevScripts.

This example shows the synchronization of all schema changes, and only changes to the static data required for testing. The transactional data in the table WidgetPurchases will not be included.

Note that:

- Scripts folders and the command line interface are only available with the SQL Compare and SQL Data Compare Professional Editions.
- Data synchronization may fail if the schemas are not identical.
Schema synchronization must therefore be performed first.

This article specifically refers to the command line syntax used in version 8 of both SQL Compare and SQL Data Compare.

To keep the database in source control, you first need to create a scripts folder. You do this using either the command line or graphical user interfaces of SQL Compare and SQL Data Compare.

This example uses the command line.

To create the schema scripts: at the command prompt, navigate to the SQL Compare directory, then type:

```
sqlcompare /db1:"WidgetDev"  
    /Makescripts:"C:\Scripts Folders\WidgetDevScripts"
```

The folder is created and populated with object creation scripts representing the schema of WidgetDev. To update the folder with data scripts, navigate to the SQL Data Compare directory, then type:

```
/sqldatacompare /db1:"WidgetDev"  
    /scr2:"C:\Scripts Folders\WidgetDevScripts"  
    /Exclude:table:WidgetPurchases  
    /sync
```

The scripts folder can now be committed to source control, providing a baseline version of the database that you can develop.

Example: synchronizing development changes

You can update the source control version of the database with development changes either manually or automatically. To update manually, you would use the graphical user interface of SQL Compare and SQL Data Compare to compare and synchronize your working copy of the scripts with the development database.

Alternatively you can create a *.bat* file to automatically check for changes to the development database and synchronize them with the source control version.

You can run this batch file when significant changes are made to the database, or schedule regular synchronizations using the Windows Scheduled Task Wizard.

To do this, save the following script:

```

cd "C:\Program Files\Red Gate\SQL Compare 8"
sqlcompare /db1:WidgetDev /scr2:"C:\Scripts\WidgetDevScripts"
    /o:Default
    /sync
cd "C:\Program Files\Red Gate\SQL Data Compare 8"
sqldatacompare /db1:WidgetDev /scr2:"C:\Scripts\WidgetDevScripts"
    /o:Default
    /Exclude:table:WidgetPurchases
    /sync

```

Where:

- */db1:WidgetDev* specifies WidgetDev as the source
- */scr2:"C:\Scripts\WidgetDevScripts"* specifies WidgetDevScripts as the target
- */o:Default* specifies that the default options will be used for comparison and synchronization
- */sync* synchronizes the data sources, making WidgetDevScripts the same as WidgetDev
- */exclude:table:WidgetPurchases* excludes WidgetPurchases. All other tables will be deployed.

Checking in the changes

The *.bat* file synchronizes your working copy of the scripts folder. It is still necessary to check those changes into your source control repository.

If your source control system allows you to check in modifications from the command line interface, you can add check in instructions to the end of the script, automating the entire process.

When you compare data sources using SQL Compare and SQL Data Compare, the command line output lists their differences. You can direct this output to a text file using */v > C:\FileName.txt*

The file could then be used as part of the check in comments, providing a detailed log of changes.

For example, if you are using the Subversion source control system, you could add the following to the end of the script:

```

svn commit C:\Scripts\WidgetDevScripts
    -file C:\FileName.txt

```

The synchronization changes are committed to source control, and a list of the synchronized differences is added as a commit message.

Example: deploying to the test environment

When development changes are sufficient to trigger a build, the database is deployed. As data synchronization may fail if the schemas are not identical, the schema changes are deployed first.

This example deploys the schema and static data of WidgetDev to the testing server WidgetTest, and creates reports as part of the build process.

To automate the deployment, save the following command lines as a *.bat* file, and run it as part of the build process:

```
cd "C:\Program Files\Red Gate\SQL Compare 8"
sqlcompare /scr1:"C:\Scripts\WidgetDevScripts" /db2:WidgetTest
    /o:Default
    /Report:"C:\SchemaDiffReport.html"
    /ReportType:Interactive
    /ScriptFile:"C:\SchemaSyncScript.sql"
    /sync
cd "C:\Program Files\Red Gate\SQL Data Compare 8"
sqldatacompare /scr1:"C:\Scripts\WidgetDevScripts" /db2:WidgetTest
    /o:Default
    /Exclude:table:WidgetPurchases
    /ScriptFile:"C:\DataSyncScript.sql"
    /sync /v > C:\DataDeploy.txt
```

Where:

- */scr1:"C:\Scripts\WidgetDevScripts"* specifies WidgetDevScripts as the source
- */db2:WidgetTest* specifies WidgetTest as the target
- */o:Default* specifies that the default options will be used for comparison and synchronization
- */sync* synchronizes the data sources, making WidgetTest the same as WidgetDevScripts
- */v > "C:\SchemaDeploy.txt"* directs detailed command line output describing the schema synchronization to a file
- */v > "C:\DataDeploy.txt"* directs detailed command line output describing the data synchronization to a file
- */Report* generates a report of the schema differences and writes it to the specified file
- */ReportType* specifies the format of the report, in this case a detailed interactive HTML format
- */ScriptFile* saves a copy of the SQL script used to migrate the changes
- */Exclude:table:WidgetPurchases* excludes WidgetPurchases. All other tables will be deployed

Creating test data

These examples have discussed the deployment of existing test data. It is also possible to create realistic test data using Red Gate's [SQL Data Generator](#)

You can set up a SQL Data Generator project specifying details of the target database, what kind of data to generate, and so on.

To generate data at build time, you can run that SQL Data Generator project from the command line:

```
sqldatagenerator /Project:"C:\Deployment\TestData.sqlgen"  
                /Out:"C:\Deployment\DataGenerationReport.txt"
```

Here, *TestData.sqlgen* is a SQL Data Generator project file, and the */Out* switch writes a summary of the data generated to the file *DataGenerationReport.txt*

Automation with MSBuild, NAnt, and Powershell

The previous examples use MS-DOS batch scripting to run SQL Compare and SQL Data Compare. Although this has the benefit of being supported by all versions of Windows, it does not integrate elegantly with contemporary build systems. Checking the script files into and out of source control using the command line requires additional scripting, and lacks flexibility.

The following examples cover some more customizable and powerful technologies.

You can download the code examples used here as a .zip file (<http://downloads.red-gate.com/HelpFiles/ContinuousIntegrationSamples.zip>)

Using MSBuild

MSBuild was released with Visual Studio 2005, and is now one of the most popular build scripting systems. Using it, you can quickly get started building your application. Although MSBuild is commonly used with Team Foundation Server, it can be used by any continuous integration server, or directly from the command line interface.

For more information, see the [MSBuild documentation](#)

The simplest way to integrate SQL Compare with MSBuild is to use the built in *Exec* task to execute command lines similar to those described earlier in this article. Although the results are identical, you can take advantage of the MSBuild infrastructure, and information such as whether the build was successful, and where files have been checked out to.

The following MSBuild target executes SQL Compare, using a scripts folder as the source:

```
<Target Name="Deploy" DependsOnTargets="Build">
  <!-- Sync from scripts to Database -->
  <Exec Command='SQLCompare.exe /scripts1:"$(CheckoutPath)\SqlCompareDB"
/Server2:"$(TargetServer)" /db2:"$(TargetDatabase)" /synchronize'
      WorkingDirectory="C:\Program Files\Red Gate\SQL Compare 8"
IgnoreExitCode="true" />
</Target>
```

The target for synchronization is then set using properties at the top of your build file. The properties specify that the target server is called SQLStaged, the target database is Northwind, and the SQL Compare script folder is checked out to the directory *C:\VSSGet*

```
<PropertyGroup>
  <TargetServer>SQLStaged</TargetServer>
  <TargetDatabase>Northwind</TargetDatabase>
  <CheckoutPath>C:\VSSGet</CheckoutPath>
</PropertyGroup>
```

These options can also be specified as arguments when you execute the MSBuild script, allowing for more flexibility.

For example:

```
MSBuild DeployToTest.msbuild /p:TargetServer=SQLStaged
```

This command executes the build script, which in turn executes SQL Compare with the appropriate settings. This can be integrated into a larger build file that also handles build and deployment.

Given the power of MSBuild, many people have created additional tasks as part of the [MSBuild Community Tasks](#) project.

For example, to check out from Visual SourceSafe you could include the following snippet:

```
<VssGet UserName="Deploy"
        Password="test"
        LocalPath="$(CheckoutPath) "
        Recursive="True"
        DatabasePath="C:\Visual SourceSafe\srcsafe.ini"
        Path="$/"
        Writable="true" />
```

However, most continuous integration servers will do this for you as part of their build process, before executing your build script.

Combining these methods, you can check out the latest version of your scripts and execute SQL Compare to update your database with a single command. The same approach can be taken with SQL Data Compare. This is demonstrated below, using NAnt.

The complete script used in this example can be found in the [downloadable .zip file](#) as the file: *DeployToTest.msbuild*

Using NAnt

NAnt is a popular alternative to MSBuild, with similar syntax.

For more information, see the [NAnt home page](#)

Here, instead of synchronizing from a scripts folder, the example synchronizes from another database. This is useful if many developers are working on a shared central database, and you want to deploy its static data.

The following NAnt script could be used:

```
<exec program="c:\program files\red gate\SQL Compare 8\SQLCompare.exe"
      commandline="/f /v /server1:${sqlcompare.server1}
/serve2:${sqlcompare.server2} /database1:${sqlcompare.database1}
/database2:${sqlcompare.database2} /synchronize"
      resultproperty="execReturnCode"
      failonerror="false"/>
```

Here, there are four different properties which are set to define the synchronization:

```
<property name="sqlcompare.server1" value="."/>
<property name="sqlcompare.server2" value="SQLStaged"/>
<property name="sqlcompare.database1" value="Northwind"/>
<property name="sqlcompare.database2" value="Northwind"/>
```

The script can then be executed via NAnt with following command:

```
NAnt.exe -buildfile:nant.build
```

The complete script used in this example can be found in the [downloadable .zip file](#) as the file: *nant.build*

Using Powershell

Automated build systems solve many continuous integration problems; however there are other times when automation is useful. Powershell - a technology Microsoft are investing in heavily - is an advanced command line and task scripting tool. It allows you to write custom scripts that solve every day problems, such as synchronizing your database.

For more information, see the [Powershell getting started guide](#)

For example, the following two snippets of code are Powershell methods which synchronize databases:

1) Synchronize the schema

```
Function SqlCompare
{
    param($Server1, $Database1, $Server2, $Database2)

    $Command = '&"C:\Program Files\Red Gate\SQL Compare 8\sqlcompare.exe"
/Server1:' + $Server1 + ' /database1:' + $Database1 + ' /server2:' +
$Server2 + ' /database2:' + $Database2 + ' /synchronize'

    Invoke-Expression $Command
}
```

2) Synchronize the data

```
Function SqlDataCompare
{
    param($Server1, $Database1, $Server2, $Database2)

    $Command = '&"C:\Program Files\Red Gate\SQL Data Compare
8\sqldatacompare.exe" /Server1:' + $Server1 + ' /database1:' + $Database1 +
' /server2:' + $Server2 + ' /database2:' + $Database2 + ' /synchronize'

    Invoke-Expression $Command
}
```

The advantage of having these commands as methods is that you can store them in a Powershell script file. That file can be included in any subsequent Powershell script, or at the interactive console in order to perform the synchronization:

```
.\SqlCompare.ps1  
SqlCompare '.' 'NorthwindDev' '(local)' 'Northwind'  
SqlDataCompare '.' 'NorthwindDev' '(local)' 'Northwind'
```

The complete script used in this example can be found in the [downloadable .zip file](#) as the file: *SQLCompare.ps1*

Conclusion

This article has outlined some examples of how to include databases in a continuous integration development cycle. Databases can be maintained in source control using the scripts folder functionality of SQL Compare and SQL Data Compare, and then deployed to a live database as part of the integration process. The Red Gate tools handle the scripting and synchronization process, removing the bottleneck that has traditionally obstructed continuous integration and source control for databases. This process can be fully automated using the command line interface, either on its own, or in conjunction with technologies such as MSBuild, NAnt, and Powershell.

Many of the tools described in this article are freely available. MSBuild is distributed as part of Microsoft Visual Studio 2005, NAnt is available as a free download, as is the Subversion source control system.

All Red Gate tools are available as a fully-featured 14 day free trial.