SQL Source Control 2.2

December 15th, 2011

Note: this documentation applies to an old version of this product.

For the latest documentation, see documentation.red-gate.com

Contents

Getting started 3
Worked example: Subversion (SVN) 4
Worked example: Team Foundation Server (TFS) 13
Worked example: Mercurial 22
Linking a database to source control 31
Committing changes 34
Source controlling data 37
Getting the latest version 39
Viewing source control logs / history 40
Getting a specific version 42
Deploying a database from source control 44
Undoing changes 47
Conflicts 49
Using SVN bug IDs & TFS work items 51
Working with config files 52
Using filters to exclude objects 54
Branching and merging 58
Setting up a local Subversion repository 61
Setting up a Subversion server 63
Using the evaluation repository 67
Moving the evaluation repository to an SVN server 69
Bug reports and feedback 72

Getting started

SQL Source Control is an add-in for SQL Server Management Studio that lets you get your database into source control.

SQL Source Control therefore brings the change management and collaboration benefits of source control to database development, without affecting your workflow, or requiring new development processes.

Note: SQL Source Control is not a source control system; it allows you to store your databases in your existing source control system.

Supported source control systems

Currently, SQL Source Control supports:

- Subversion (SVN)
- Team Foundation Server (TFS)
- SourceGear Vault
- Any source control system with a command line interface. Presets are included for:
  - Mercurial
  - Git
  - Perforce

SQL Source Control: step-by-step

1. Link the database to source control (page 31)
2. Commit the objects (page 34)
3. Optionally commit static (lookup) data (page 37)
4. Make development changes normally
5. Commit changes or get the latest version (page 39)

Worked examples

Learn more about SQL Source Control by following a detailed example:

- Worked example: setting up SQL Source Control with SVN (page 4)
- Worked example: setting up SQL Source Control with TFS (page 13)
- Worked example: setting up SQL Source Control with Mercurial (page 22)
- Worked example: Deploying with migration scripts

Worked example: Subversion (SVN)

This example shows you how to set up database source control so teams of developers can work on a database update.

This example uses:

- The Subversion (http://subversion.apache.org/) source control system
- The Tortoise SVN (http://tortoisesvn.tigris.org/) client for Subversion

To follow this example, you should download and install the latest version of TortoiseSVN (http://tortoisesvn.net/downloads)

In the example, the Magic Widget Company has a SQL Server database running on a live web server. This database contains a number of tables, views, stored procedures, and other database objects. The Magic Widget Company's development team wants to begin working on an update to this database. They have already created a copy of the production database, as a baseline to develop against.

They now need to source control the development database, so that each developer can get their own dedicated copy to work on.

This example has 6 steps:

1. Set up the database (page 4)
2. Link the database to source control (page 4)
3. Commit the database objects to source control (page 7)
4. Get the latest version (page 9)
5. Make development changes (page 10)
6. Commit the development changes to source control (page 11)

1. Set up the database

This worked example uses the *WidgetDevelopment* database.

To create the database on your SQL Server:

1. If it already exists, delete *WidgetDevelopment*
2. Click here to view the SQL creation script (/support/SQL_Source_Control/help/2.0/WidgetDevScript.sql) for the database.
3. Copy the script, paste it into a query window in SQL Server Management Studio, and run it.

The database and its schema are created.

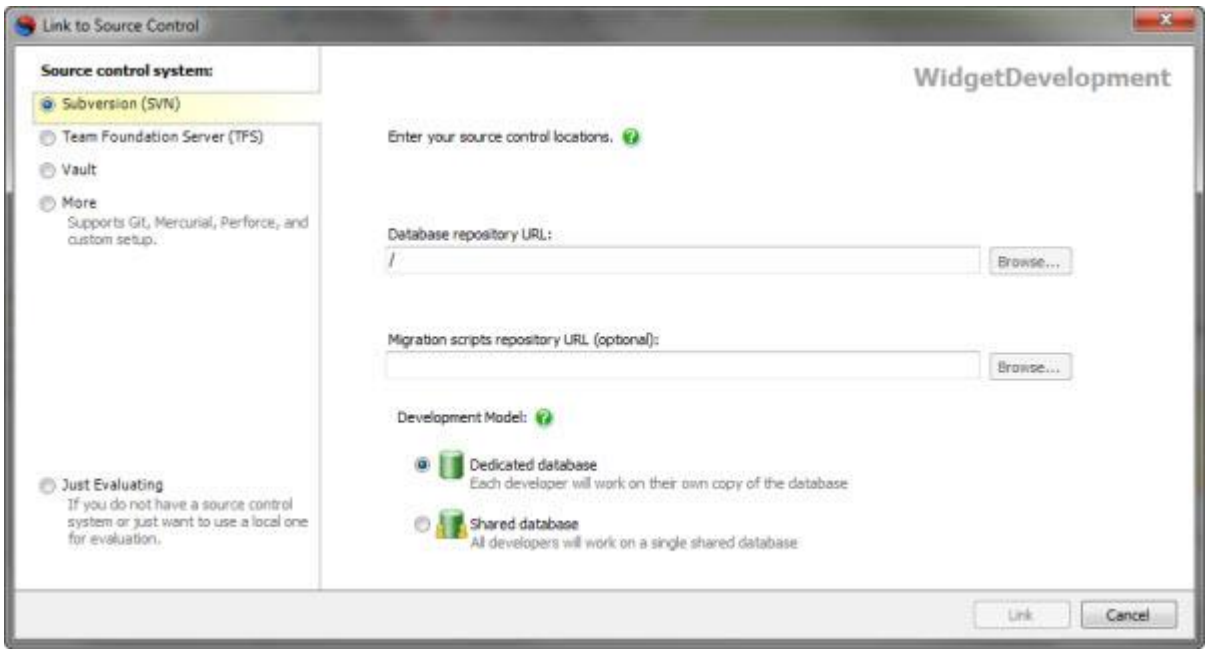
2. Link the database to source control

Linking associates the database with a location in source control.

To link the database:

1. Open SQL Server Management Studio if it is not already open.
2. In the **Object Explorer**, select the *WidgetDevelopment* database, right-click, and click **Link database to source control**

   The Link to Source Control dialog box is displayed:

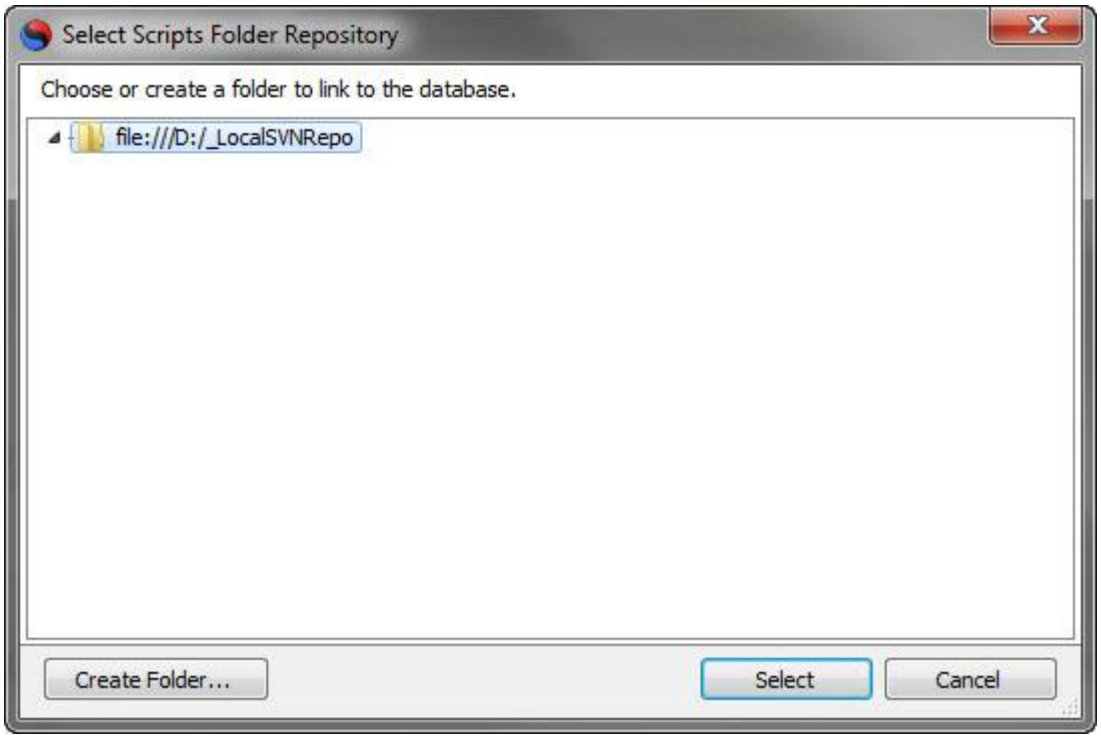


3. Under **Source control system**, on the left hand side, ensure *Subversion* is selected.
4. In **Repository URL,** type or paste the URL for the root of your source control repository. For example: *https://WidgetSourceCode.MagicWidget.com/*

   The **Browse** button is enabled.

   Note that the repository URL is case sensitive.
5. Click **Browse**

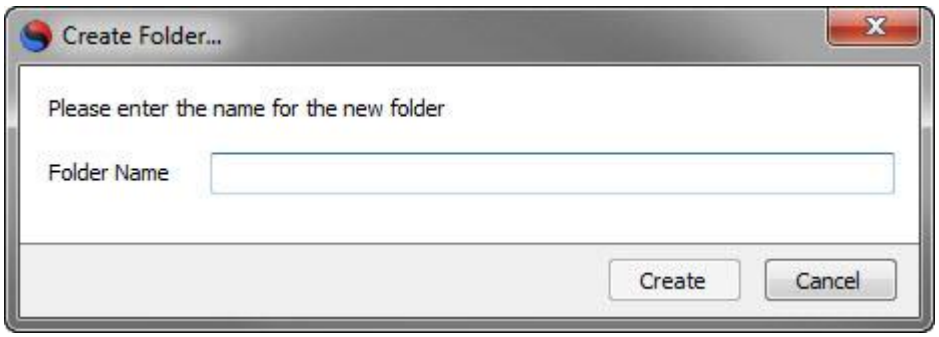
The Select Scripts Folder Repository dialog box is displayed:



Here you can select an folder to link to, or create one if you are linking for the first time. We will create a folder.

6. Click **Create Folder**

   The Create Folder dialog box is displayed:



   Type a name for the folder.

   In this example, name the folder: *WidgetDevScripts*

7. Click **Create**

   A dialog box is displayed for you to supply a commit comment for the creation of the new folder in source control. Type a comment, and click **OK**

   The folder is created, and appears in the list on the Select Scripts Folder Repository dialog box.

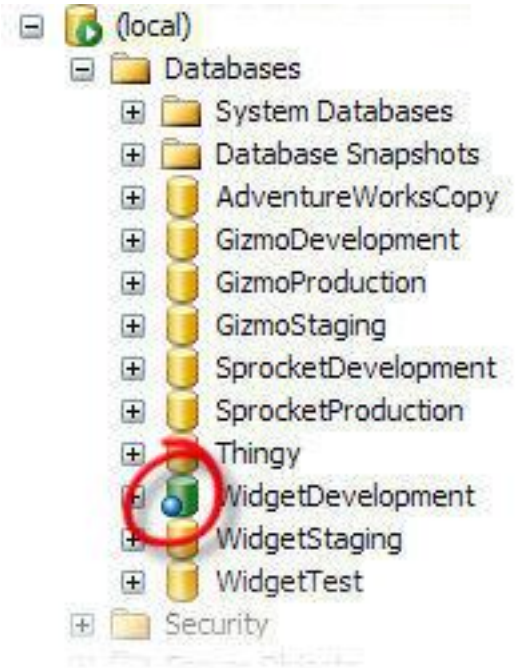
8. Ensure the folder you created is selected, and click **Select**

9. On the Link to Source Control dialog box, the **Link** button is now enabled.
10. Click **Link**

    You may be prompted for login credentials for your source control repository.

    A link to the repository is created, and SQL Source Control is now able to determine differences between the database and the repository.

    The database icon in the Object Explorer changes, showing that the database is linked to source control, and that there are changes to commit:



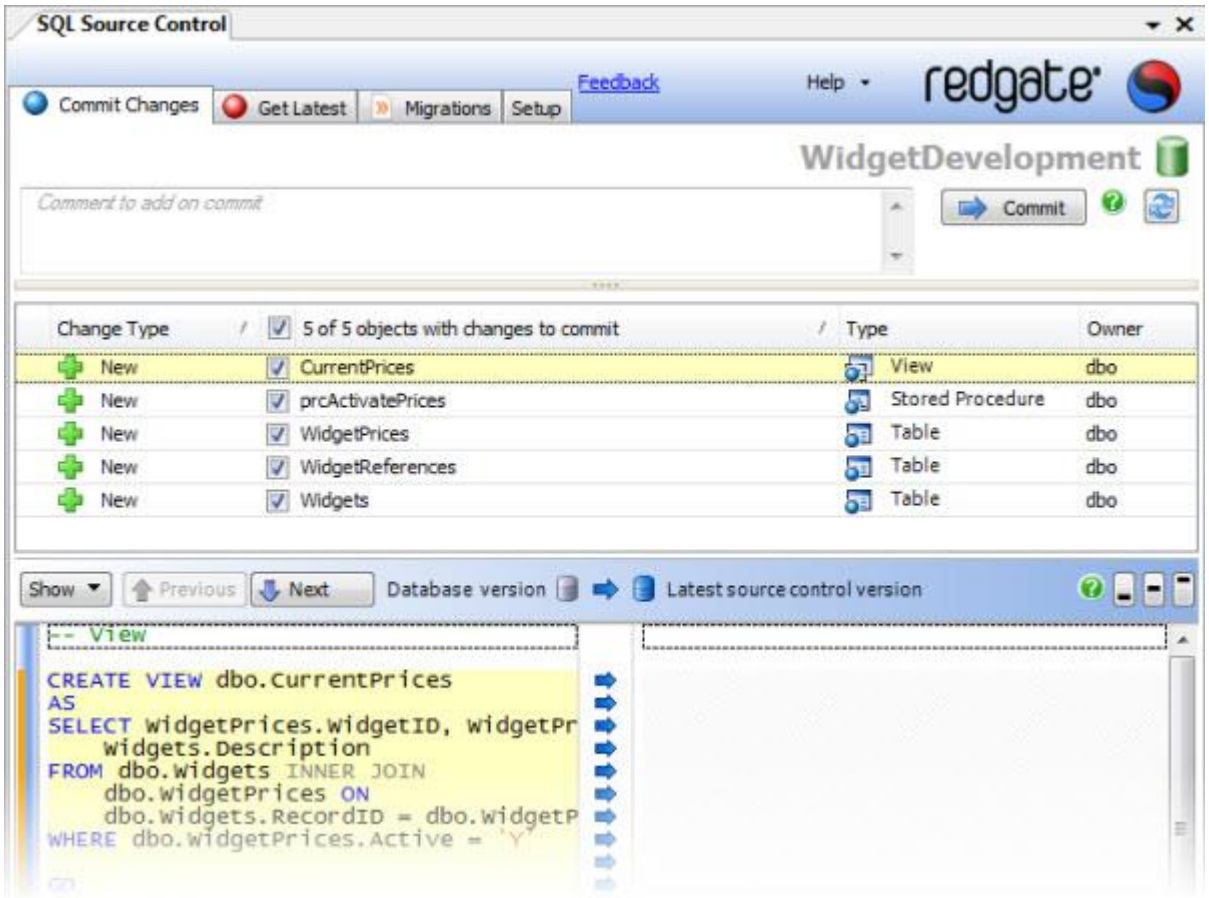
    Note that the database objects have not yet been committed to source control.

3. Commit the database objects to source control

To finish source controlling the database, commit the objects:

1. In SQL Source Control, click the **Commit Changes** tab.

The Commit Changes tab displays a list of all the objects with database versions that do not match the latest source control version:



Because none of the objects yet have versions in source control, they are all listed.

You can view the creation script for an object by clicking it. The script is displayed in the Object Differences pane, below the list of objects to commit.

2. In **Comment to add on commit**, type or paste a comment.

   Comments are recommended as they help provide a detailed change history.

   In this example, type *Initial commit of all objects*.

3. Click **Commit**

   SQL Source Control displays a message dialog box showing the progress of the commit.

   When the commit is complete, click **OK** to close the message box.

The objects are committed to source control and other users can now get the latest version of the database.

The Object Explorer is updated to show that there are now no outstanding changes to commit.

4. Get the latest version

Now the database is in source control, another user can get the latest version, and make development changes.

They create a new database, link it to source control, and update it with the latest version of all the objects.

Linking the database

1. In SQL Server Management Studio, create a new database, with the name *WidgetDevelopment*
2. In the **Object Explorer**, ensure *WidgetDevelopment* is selected. On the **Setup** tab, click **Link database to source control**.

   The Link Database to Source Control dialog box is displayed.
3. Under **Source control system**, on the left hand side, ensure *Subversion* is selected.
4. In **Repository URL**, type or paste the URL of the database in source control.

   Note that the repository URL is case sensitive.
5. Click **Link**

   You may be prompted for login credentials for your source control repository.

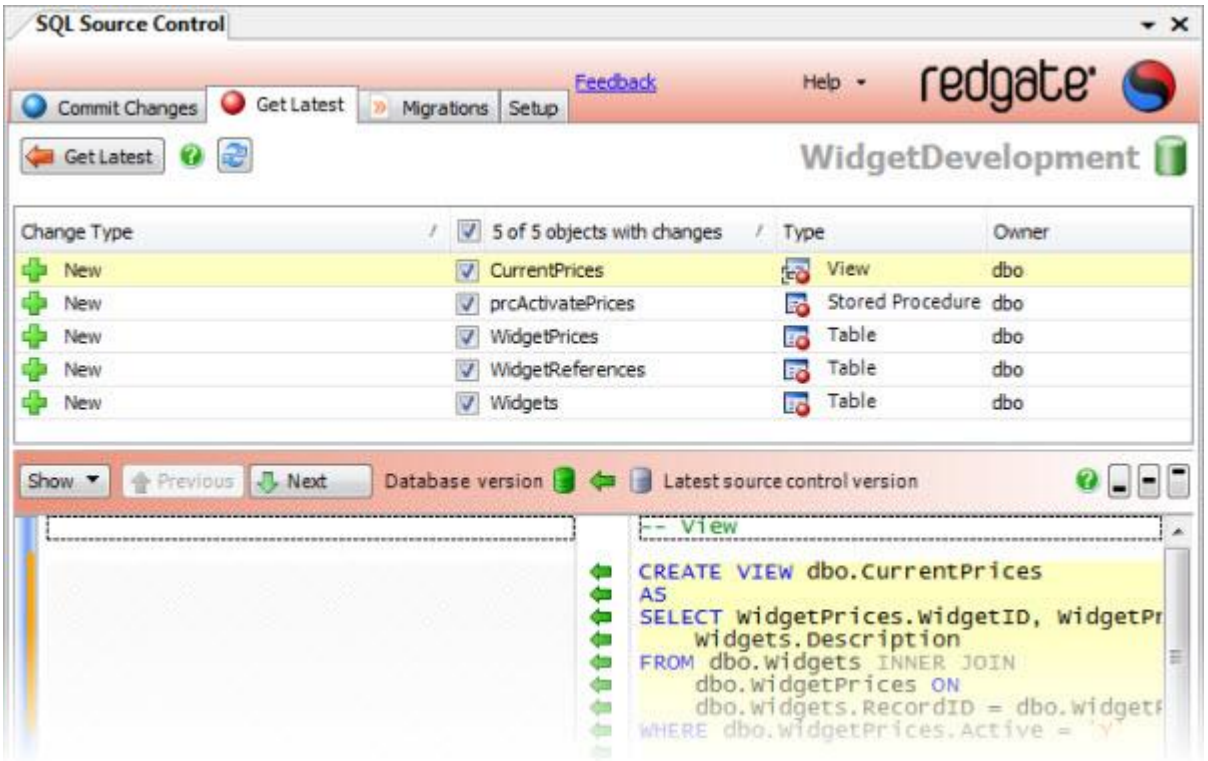
   A link to the repository is created, and SQL Source Control is able to determine differences between the database and the repository.

   However, the database has not yet been updated with the objects from source control.

Getting the latest version

1. Click the **Get Latest** tab.

Because you do not yet have any of the objects in your database, all the objects in *WidgetDevelopment* are listed here:



2. Ensure all of the objects are selected.
3. Click **Get Latest**

   A progress dialog box is displayed while SQL Source Control updates the database.

   The database is updated to the latest version.

5. Make development changes

Development proceeds normally, and the database is modified.

A column is changed in the table *Widgets*, to allow longer widget descriptions. This change is committed to source control.

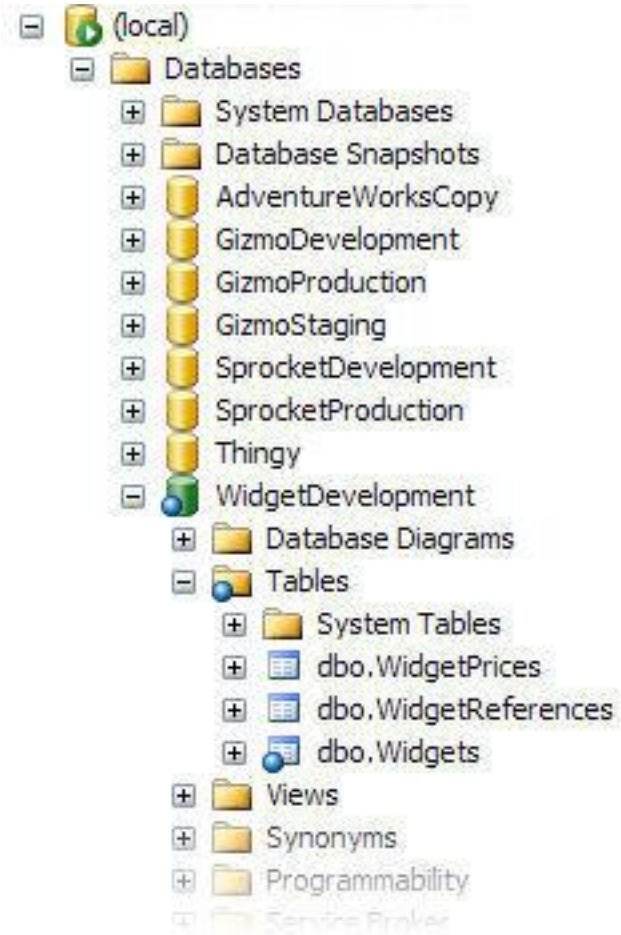
1. In SQL Server Management Studio, open a new query window, and type or paste the following SQL statement:

```
USE WidgetDevelopment
GO
ALTER TABLE [dbo].[Widgets] ALTER COLUMN [Description] [nvarchar] (100)
```

2. Click **Execute** or press **F5**

   The script runs; the database is updated.

3. SQL Source Control detects the change to the database, and highlights the affected object in the Object Explorer:

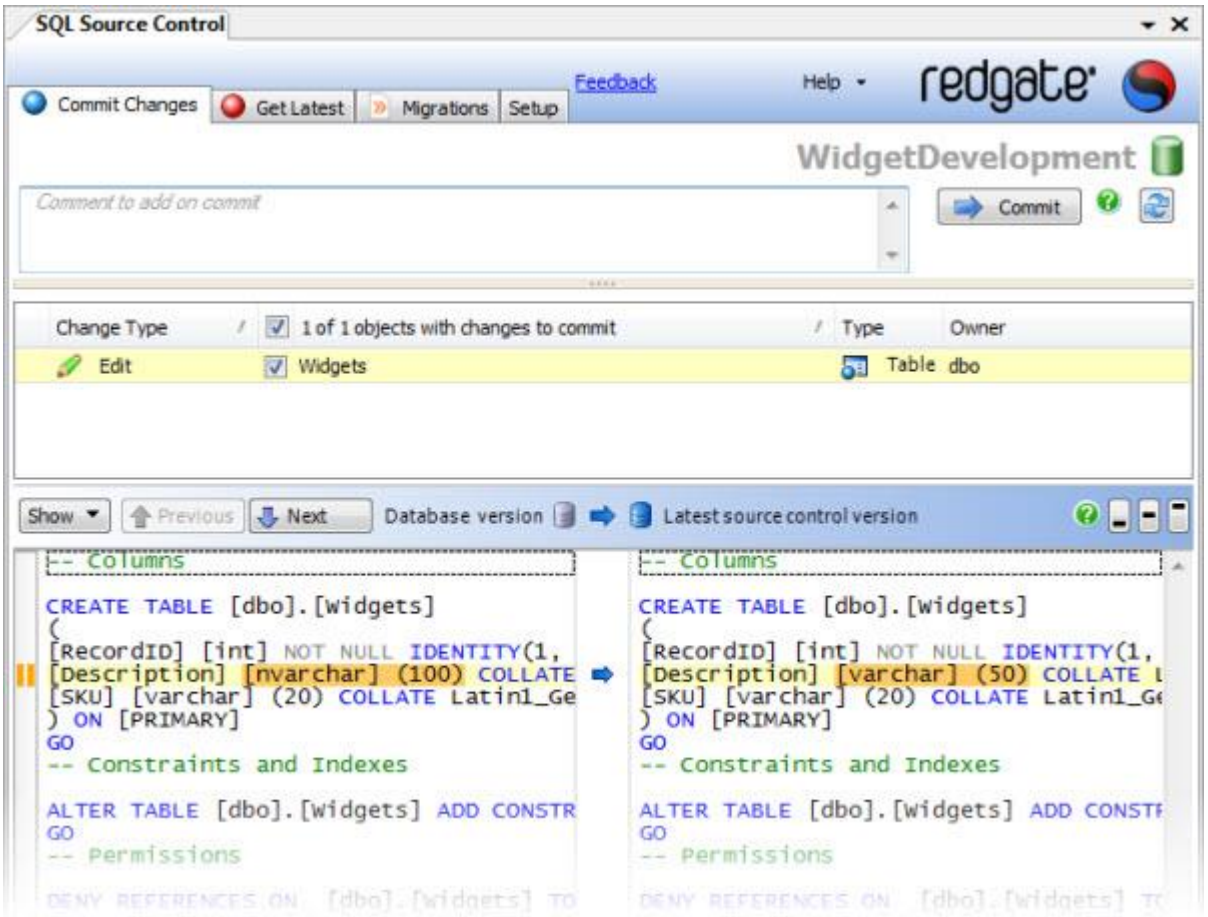


You can now commit the change.

6. Commit the development changes to source control

1. In the Object Explorer, right click the table *Widgets,* and click **Commit changes to source control**

The Commit changes tab is displayed:



The change to the table *Widgets* is listed as an edit you can commit.

The Object Differences pane shows the difference between your database version, and the latest version in source control.

2. In **Comment to add on commit**, type or paste a comment.

   In this example, type *Modified Description column*

3. Click **Commit**

   SQL Source Control displays a message dialog box that shows the progress of the commit.

   When the commit is complete, click **OK** to close the message box.

Your change is committed to source control. The Commit Changes tab lists no objects with changes to commit, and no objects are highlighted in the Object Explorer.

Other users can now get the latest version of the table.

Worked example: Team Foundation Server (TFS)

This example shows you how to set up database source control so that teams of developers can work on a database update.

Note: If you are using Team Foundation Server 2012 or tfspreview.com, you must first edit a config file. For information on how to do this, see Using SQL Source Control with Team Foundation Server 2012 or tfspreview.com (/SupportCenter/Content/SQL_Source_Control/knowledgebase/SoC_Using_TFS2012).

This example requires:

- The Team Foundation Server (TFS) source control system
- Microsoft Team Explorer client for Visual Studio

In the example, the Magic Widget Company has a SQL Server database. This database contains a number of tables, views, stored procedures, and other database objects. The Magic Widget Company's development team wants to begin working on an update to this database. They have already created a copy of the production database, as a baseline to develop against.

They now need to source control the development database, so that each developer can work on their own dedicated copy.

This example has 6 steps:

1. Set up the database (page 4)
2. Link the database to source control (page 14)
3. Commit the database objects to source control (page 7)
4. Get the latest version
5. Make development changes (page 10)
6. Commit the development changes to source control (page 11)

1. Set up the database

This worked example uses the *WidgetDevelopment* database.

To create the database on your SQL Server:

1. If it already exists, delete *WidgetDevelopment*
2. Click here to view the SQL creation script (/support/SQL_Source_Control/help/2.0/WidgetDevScript.sql) for the database.
3. Copy the script, paste it into a query window in SQL Server Management Studio, and run it.

The database and its schema are created.

2. Link the database to source control

Linking associates the database with a location in source control.

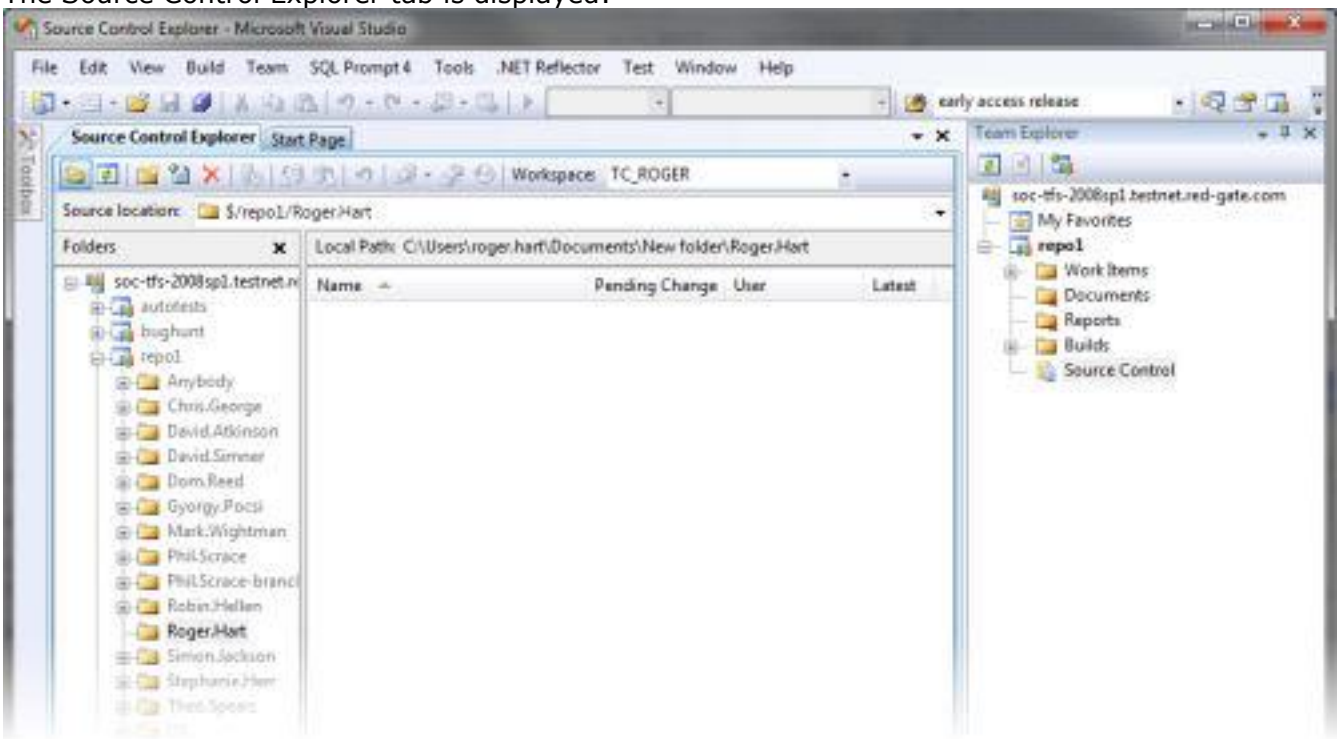
That location must be an existing, empty folder.

We will create a folder and link the database to that location.

Create a folder in source control

1. Open Visual Studio if it is not already running, and in the **Team Explorer** pane, under the server you are using, double-click **Source Control**

   The Source Control Explorer tab is displayed:



2. In the Folders pane, browse to your source control location.
3. In the file list, right-click, and click **New Folder**

   A folder is created. Name it *WidgetDev*

   Note that in some circumstances, the option to create a new folder is not available.

   If this occurs, in the Folders pane, right click your source control location, and click **Get Latest**.

   Once the local copy has updated, the option to create a folder becomes available.
4. Right-click *WidgetDev*, and click **Check In Pending Changes**

   The Check In dialog box is displayed.
5. Type a comment, and click **Check in**

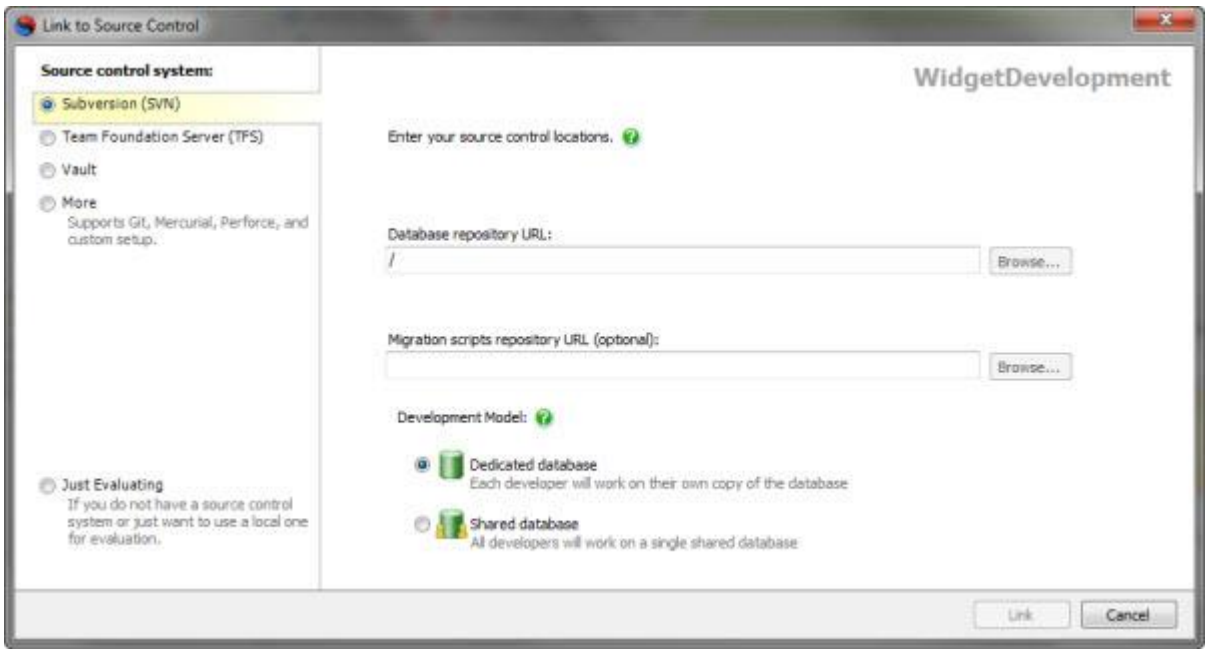
   The folder is committed to source control.

Create a link to source control

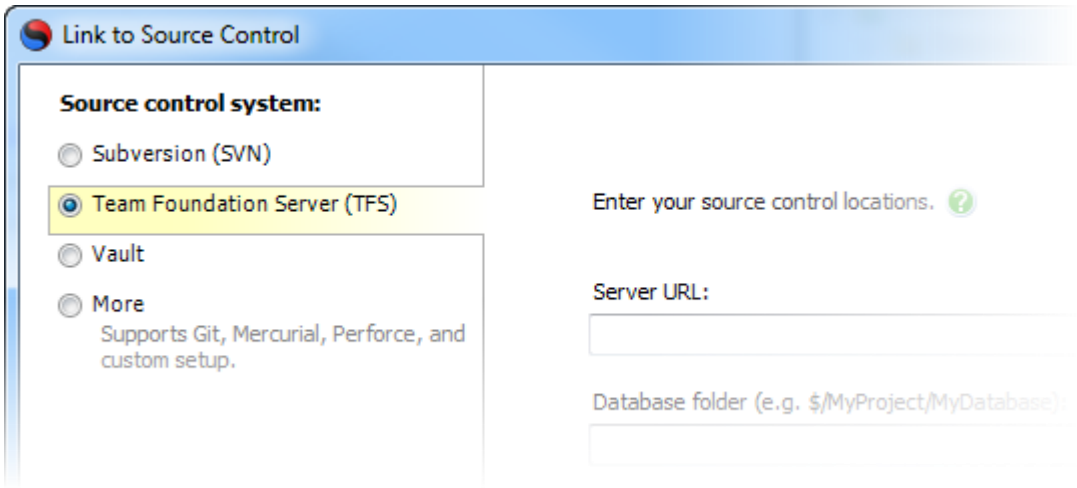
1. Open SQL Server Management Studio if it is not already open.
2. In the **Object Explorer**, select the *WidgetDevelopmen*t database, right-click, and click **Link database to source control**

   The SQL Source Control Setup tab is displayed.
3. Click **Create new link to source control**

   The Create Link to Source Control dialog box is displayed:



4. Under **Source control system**, on the left-hand side, ensure *Team Foundation Server* is selected:



5. In **Server URL,** type or paste the URL for the server you are using if it is not already filled in.

6. In **Source Control Folder**, type or paste the location of the WidgetDev folder you created.

   Alternatively, click **Browse** and browse to the location of the folder.

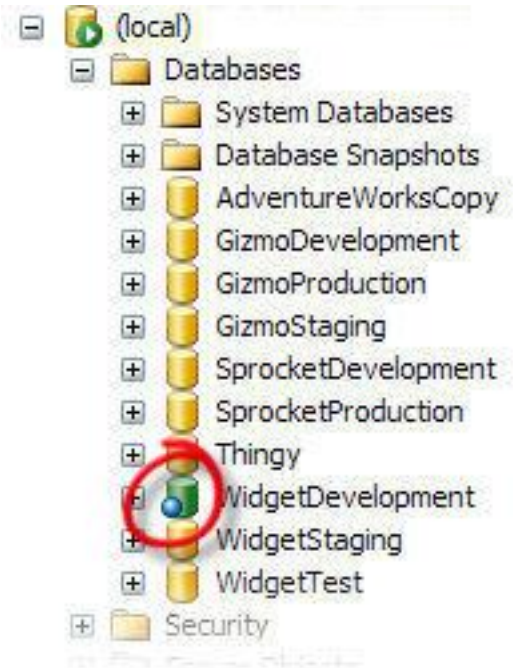
7. Under Development Model, ensure **Dedicated** is selected.

8. Click **Create Link**

   You may be prompted for login credentials for your source control repository.

   A link to source control is created, and SQL Source Control is now able to determine differences between the database and source control.

   The database icon in the Object Explorer changes, indicating that the database is linked to source control, and that there are changes to commit:



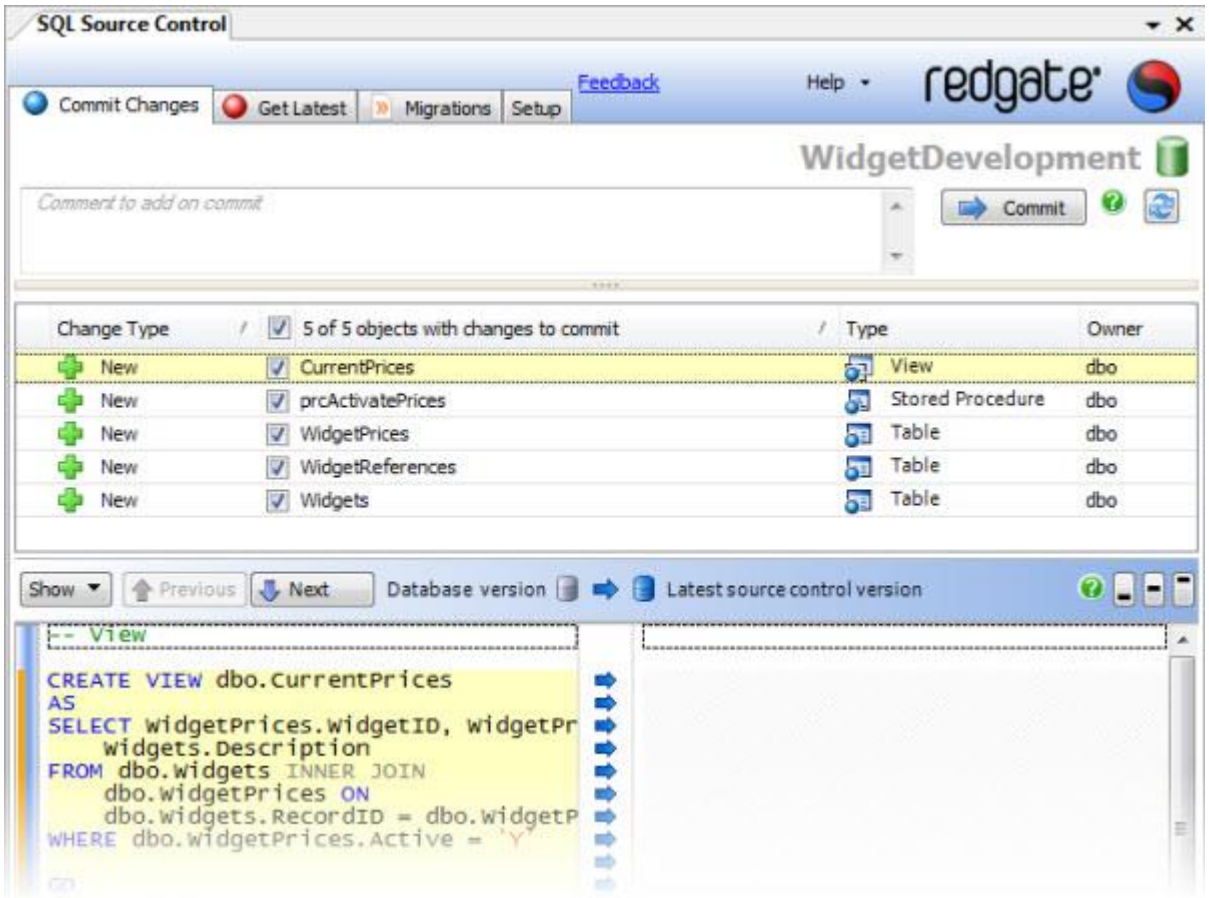
   Note that the database objects have not yet been committed to source control.

3. Commit the database objects to source control

To finish source controlling the database, commit the objects:

1. In SQL Source Control, click the **Commit Changes** tab.

The Commit Changes tab displays a list of all the objects with database versions that do not match the latest source control version:



Because none of the objects yet have versions in source control, they are all listed.

You can view the creation script for an object by clicking it. The script is displayed in the Object Differences pane, below the list of objects to commit.

2. In **Comment to add on commit**, type or paste a comment.

   Comments are recommended as they help provide a detailed change history.

   In this example, type *Initial commit of all objects*.

3. Click **Commit**

   SQL Source Control displays a message dialog box showing the progress of the commit.

   When the commit is complete, click **OK** to close the message box.

The objects are committed to source control and other users can now get the latest version of the database.

The Object Explorer is updated to show that there are now no outstanding changes to commit.

4. Get the latest version

Now the database is in source control, another user can get the latest version, and begin making development changes.

They create a new database, link it to source control, and update it with the latest version of all the objects.

Linking the database

1. In SQL Server Management Studio, create a new database, and call it *WidgetDevelopment*
2. In the **Object Explorer**, ensure *WidgetDevelopment* is selected. On the **Setup** tab, click **Link to a database already in source control**.

   The Link Database to Source Control dialog box is displayed.
3. Under **Source control system**, on the left hand side, ensure *Team Foundation Server* is selected.
4. In **Server URL**, type or paste the URL for the server you are using if it is not already filled in.
5. In **Source Control Folder**, type or paste the location of the WidgetDev folder.

   Alternatively, click **Browse** and browse to the location of the folder.
6. Click **Link**

   You may be prompted for login credentials for your source control repository.

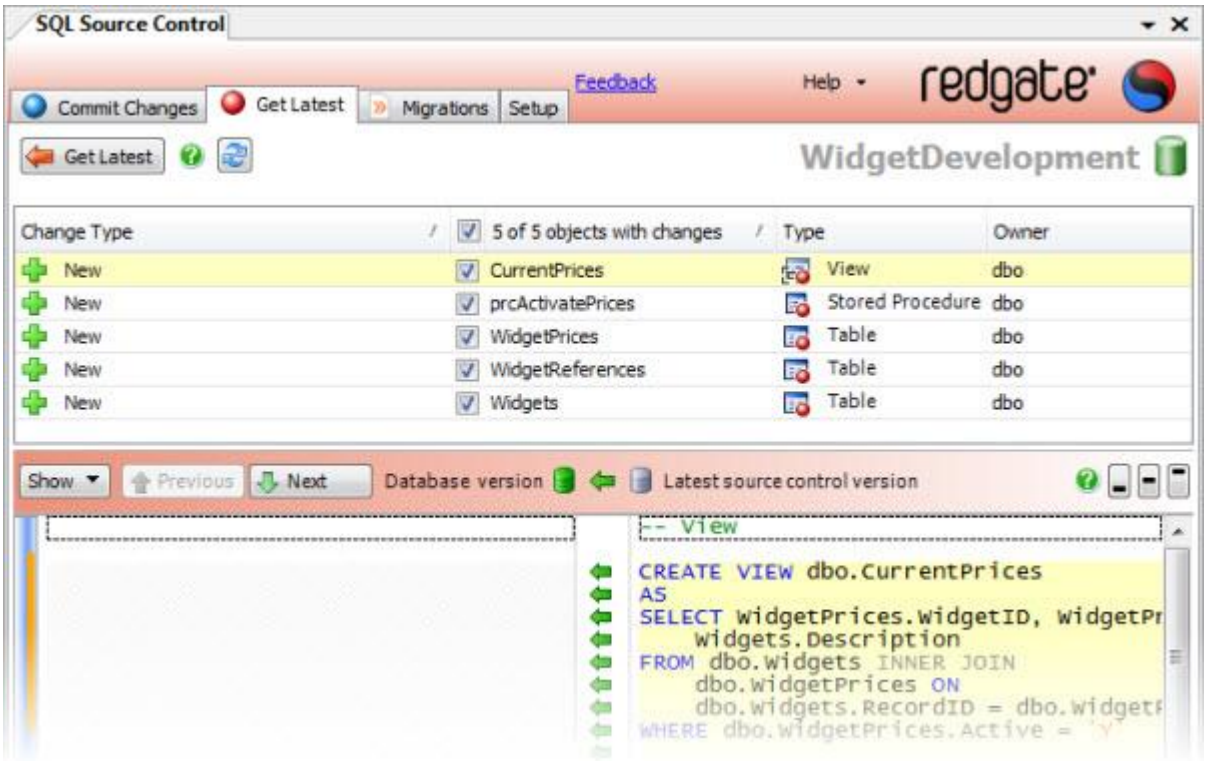
   A link to the repository is created, and SQL Source Control is able to determine differences between the database and the repository.

   However, the database has not yet been updated with the objects from source control.

Getting the latest version

1. Click the **Get Latest** tab.

Because you do not yet have any of the objects in your database, all the objects in *WidgetDevelopment* are listed here:



2. Ensure all of the objects are selected.
3. Click **Get Latest**

   A progress dialog box is displayed while SQL Source Control updates the database.

   The database is updated to the latest version.

5. Make development changes

Development proceeds normally, and the database is modified.

A column is changed in the table *Widgets*, to allow longer widget descriptions. This change is committed to source control.

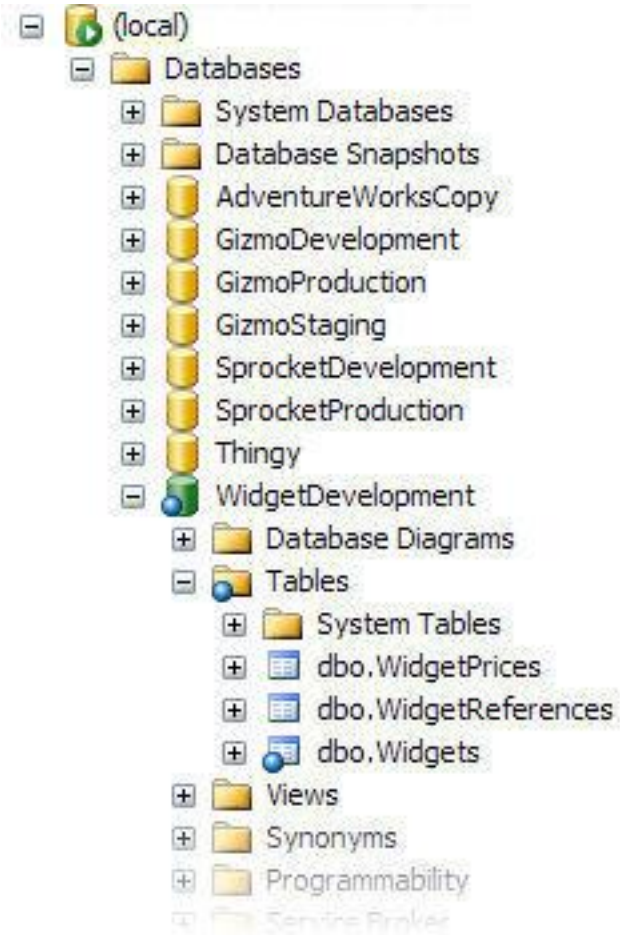
1. In SQL Server Management Studio, open a new query window, and type or paste the following SQL statement:

```
USE WidgetDevelopment
GO
ALTER TABLE [dbo].[Widgets] ALTER COLUMN [Description] [nvarchar] (100)
```

2. Click **Execute** or press **F5**

   The script runs; the database is updated.

3. SQL Source Control detects the change to the database, and highlights the affected object in the Object Explorer:

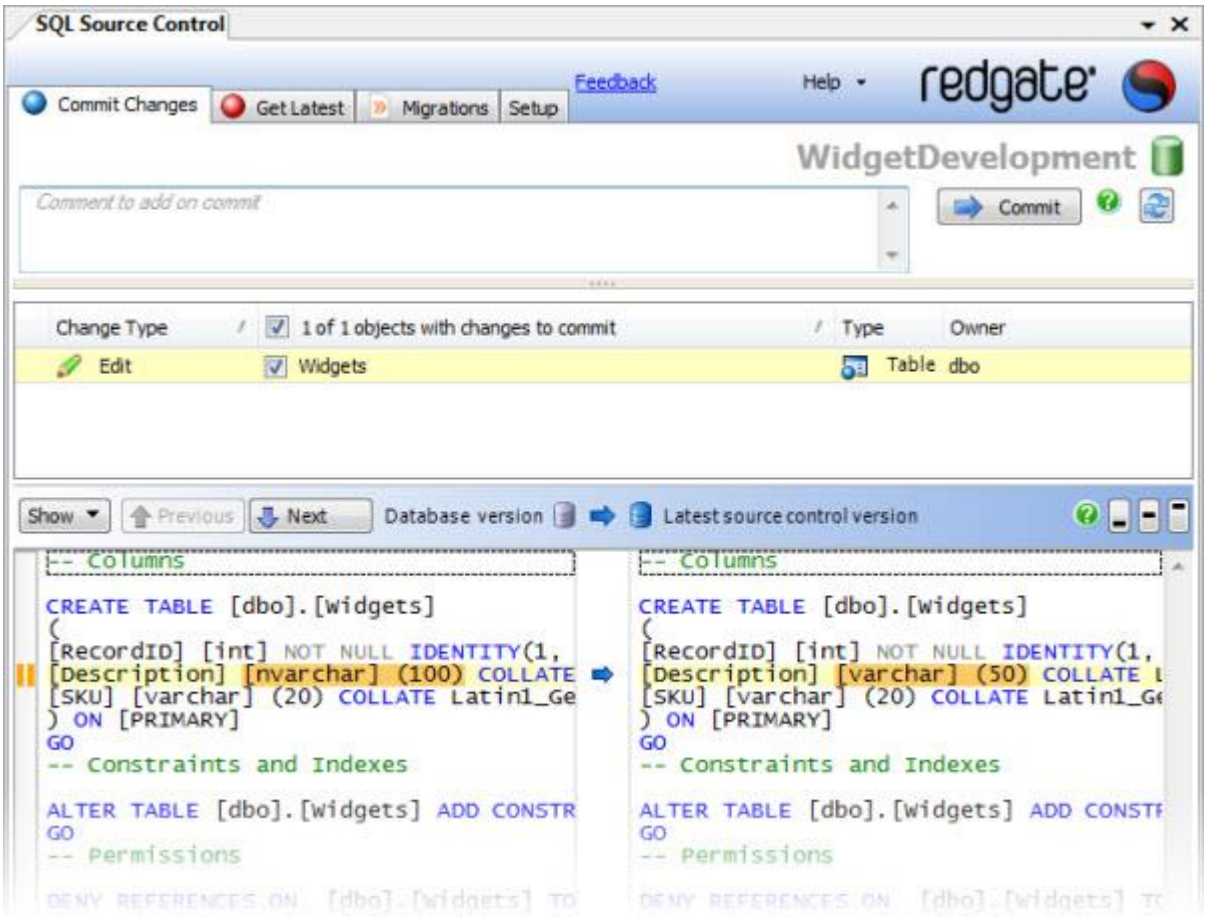


You can now commit the change.

6. Commit the development changes to source control

1. In the Object Explorer, right click the table *Widgets,* and click **Commit changes to source control**

The Commit changes tab is displayed:



The change to the table *Widgets* is listed as an edit you can commit.

The Object Differences pane shows the difference between your database version, and the latest version in source control.

2. In **Comment to add on commit**, type or paste a comment.

   In this example, type *Modified Description column*

3. Click **Commit**

   SQL Source Control displays a message dialog box that shows the progress of the commit.

   When the commit is complete, click **OK** to close the message box.

Your change is committed to source control. The Commit Changes tab lists no objects with changes to commit, and no objects are highlighted in the Object Explorer.

Other users can now get the latest version of the table.

Worked example: Mercurial

You can use SQL Source Control with any source control system with a command line interface. This example demonstrates how to set up using SQL Source Control with the Mercurial source control system.

The example uses:

- The Mercurial (http://mercurial.selenic.com/) source control system
- The TortoiseHg (http://tortoisehg.bitbucket.org/) client for Mercurial

In the example, the Magic Widget Company has a SQL Server database. This database contains a number of tables, views, stored procedures, and other database objects. The Magic Widget Company's development team wants to begin working on an update to this database. They have already created a copy of the production database, as a baseline to develop against.

They now need to source control the development database, so that each developer can work on their own dedicated copy.

This example has 6 steps:

1. Set up the database (page 4)
2. Link the database to source control (page 22)
3. Commit the database objects to source control (page 7)
4. Get the latest version (page 27)
5. Make development changes (page 10)
6. Commit the development changes to source control (page 11)

1. Set up the database

This worked example uses the *WidgetDevelopment* database.

To create the database on your SQL Server:

1. If it already exists, delete *WidgetDevelopment*
2. Click here to view the SQL creation script (/support/SQL_Source_Control/help/2.0/WidgetDevScript.sql) for the database.
3. Copy the script, paste it into a query window in SQL Server Management Studio, and run it.

The database and its schema are created.

2. Link the database to source control

Linking associates the database with a location in source control.

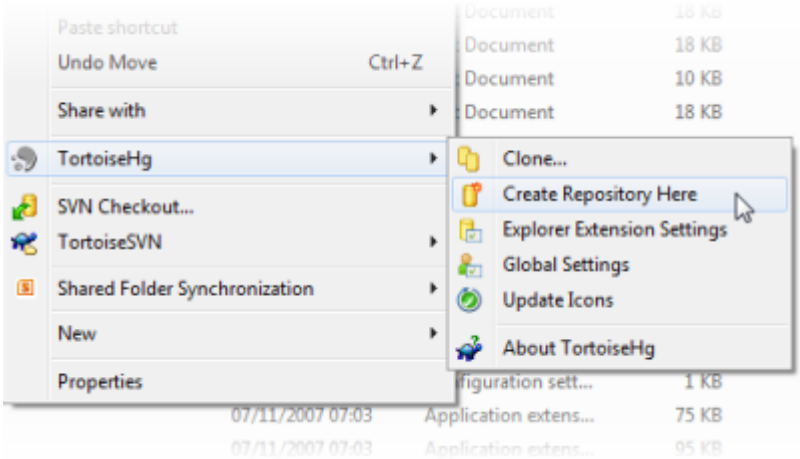
That location must be an existing, empty folder.

We will create a working folder and link the database to that location.

Create a working folder

To create an empty Mercurial working folder:

1. In a Windows Explorer window, right-click, and from TortoiseHg, select **Create Repository Here**:



2. Specify a location for the repository, and click **Create**.

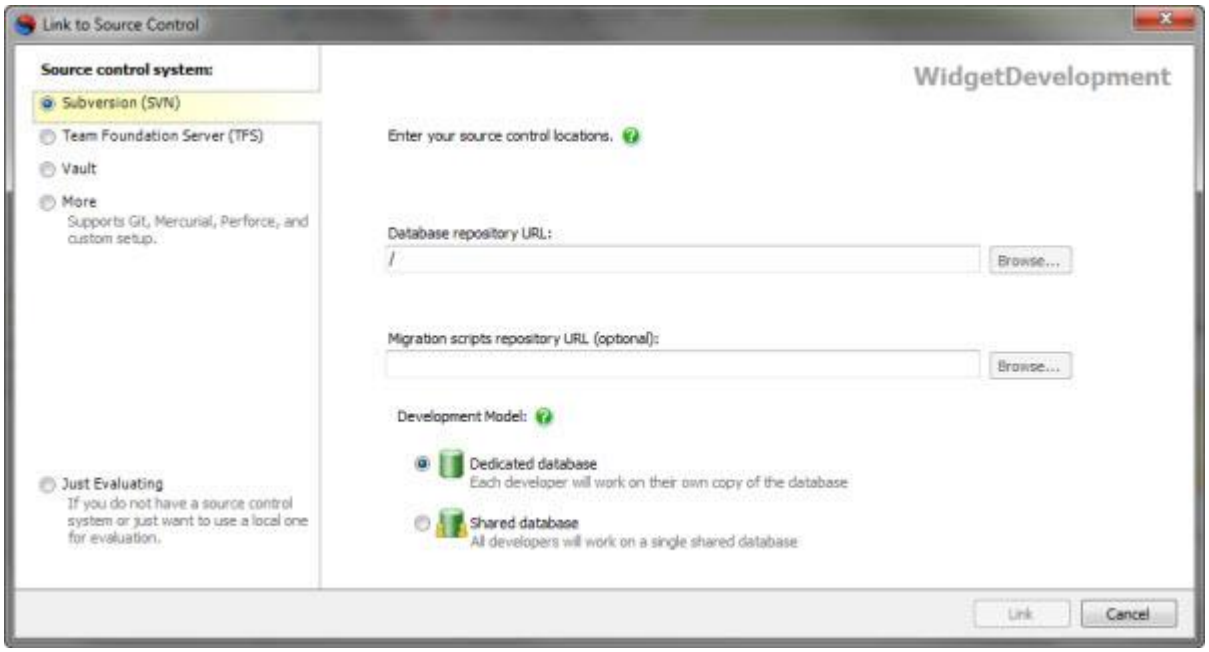
Alternatively, in the Mercurial command line interface, create a folder, navigate to it, and type `hg init`.

If you are linking to a database already in source control, you first need to get the latest version to a local working folder.

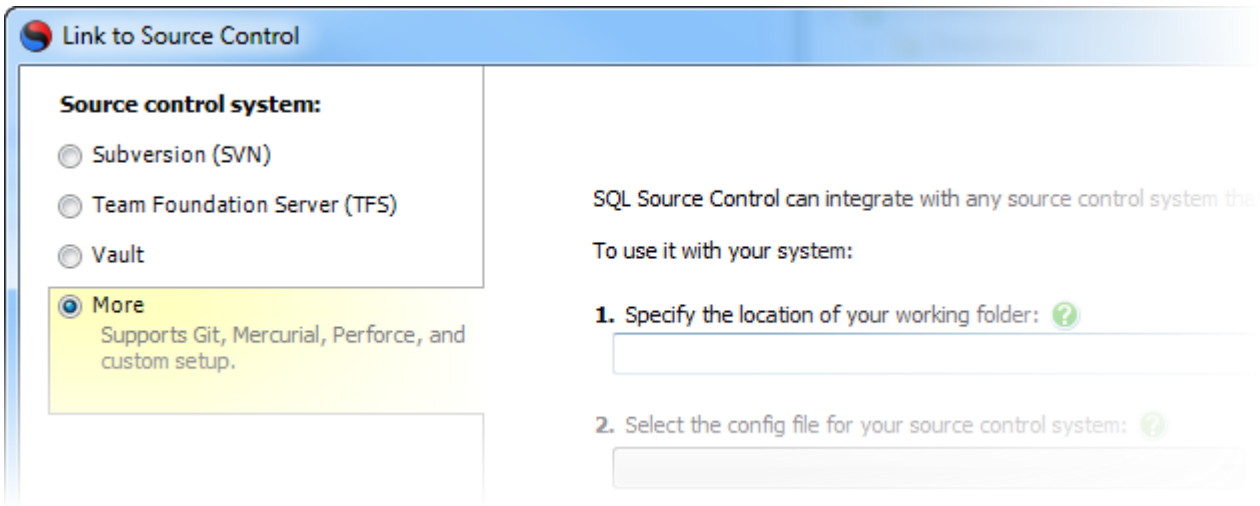
Create a link to source control

1. In SQL Source Control, on the **Setup** tab, ensure a database is selected, and click **Link database to source control**.

The Link to Source Control dialog box is displayed:



2. Under **Source control system**, on the left-hand side, select **More**:



3. Type, or browse to, the location of your working folder.
4. Select the config file for your source control system.

   In this example, select **Mercurial**.

   A config file is an XML file containing command line hooks, that let you automate source control operations (Add, Edit, Delete, etc) for different source control systems. For information on the commands in the config file, see Working with config files (page 52)

   When you first link a database to source control, the config file you select is committed to the working folder. Note that:

- if you make any changes to a config file in a working folder, you must commit the changes using your source control system.
- if you remove the config file from your working folder, you may encounter errors using SQL Source Control.

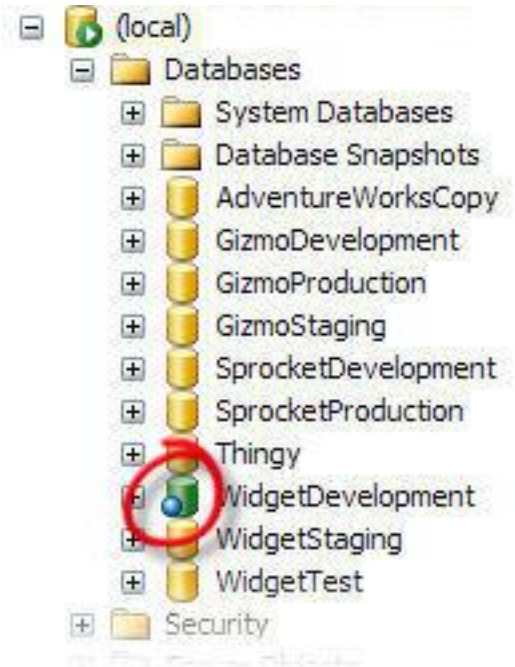
5. Select a development model.

   For more information, see Database development models (http://www.red-gate.com/supportcenter/Content.aspx?p=SQL%20Source%20Control&c=SQL_Source_Control\articles\SSC_Development_Models.htm)

6. Click **Link**.

   The database is linked to source control.

   The database icon in the Object Explorer changes to show that the database is linked:



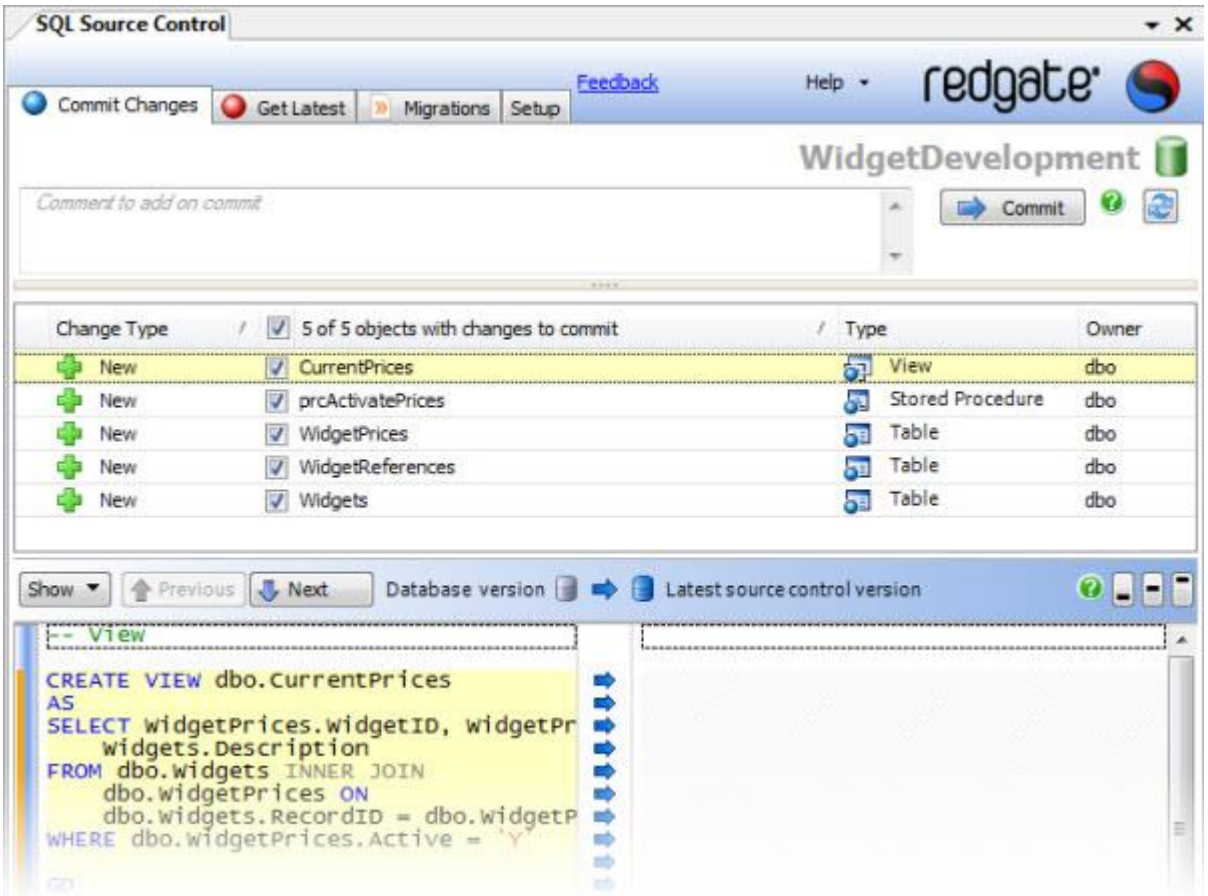
Note that linking only associates the database with a location in source control.

3. Commit the database objects to source control

To finish source controlling the database, commit the objects:

1. In SQL Source Control, click the **Commit Changes** tab.

The Commit Changes tab displays a list of all the objects with database versions that do not match the latest source control version:



Because none of the objects yet have versions in source control, they are all listed.

You can view the creation script for an object by clicking it. The script is displayed in the Object Differences pane, below the list of objects to commit.

2. In **Comment to add on commit**, type or paste a comment.

   Comments are recommended as they help provide a detailed change history.

   In this example, type *Initial commit of all objects*.

3. Click **Commit**

   SQL Source Control displays a message dialog box showing the progress of the commit.

   When the commit is complete, click **OK** to close the message box.

The objects are committed to source control and other users can now get the latest version of the database.

The Object Explorer is updated to show that there are now no outstanding changes to commit.

4. Get the latest version

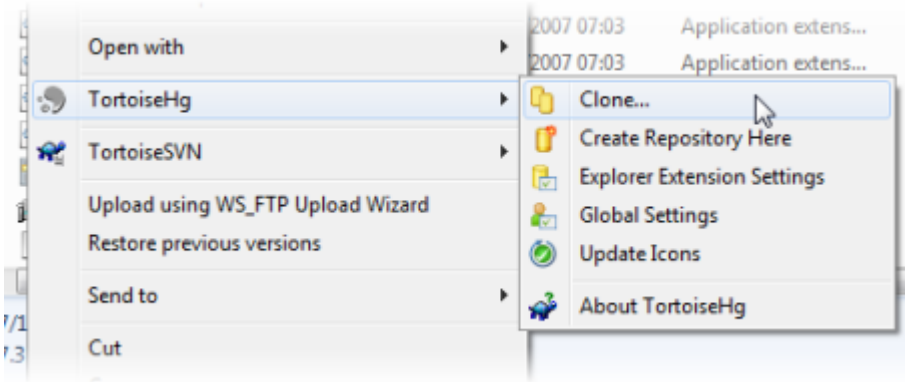
Now the database is in source control, another user can get the latest version, and make development changes.

They create a new database, link it to source control, and update it with the latest version of all the objects.

Creating a clone of the database in source control

Before you can link the database, you need to get the latest version to a local working folder. You can do this by creating a clone of the database already in source control:

1. In a Windows Explorer window, right-click, and from TortoiseHg, select **Clone**:



2. Specify the location of the database working folder in source control, and a location for the local working folder.
3. Click **Clone**.

Linking the database

1. In SQL Server Management Studio, create a new database, with the name *WidgetDevelopment*
2. In the **Object Explorer**, ensure *WidgetDevelopment* is selected. On the **Setup** tab, click **Link database to source control**

   The Link Database to Source Control dialog box is displayed.
3. Under **Source control system**, on the left hand side, ensure *More* is selected.
4. Type, or browse to, the location of the local working folder.

   The Mercurial config file in the working folder is selected automatically.
5. Click **Link**.

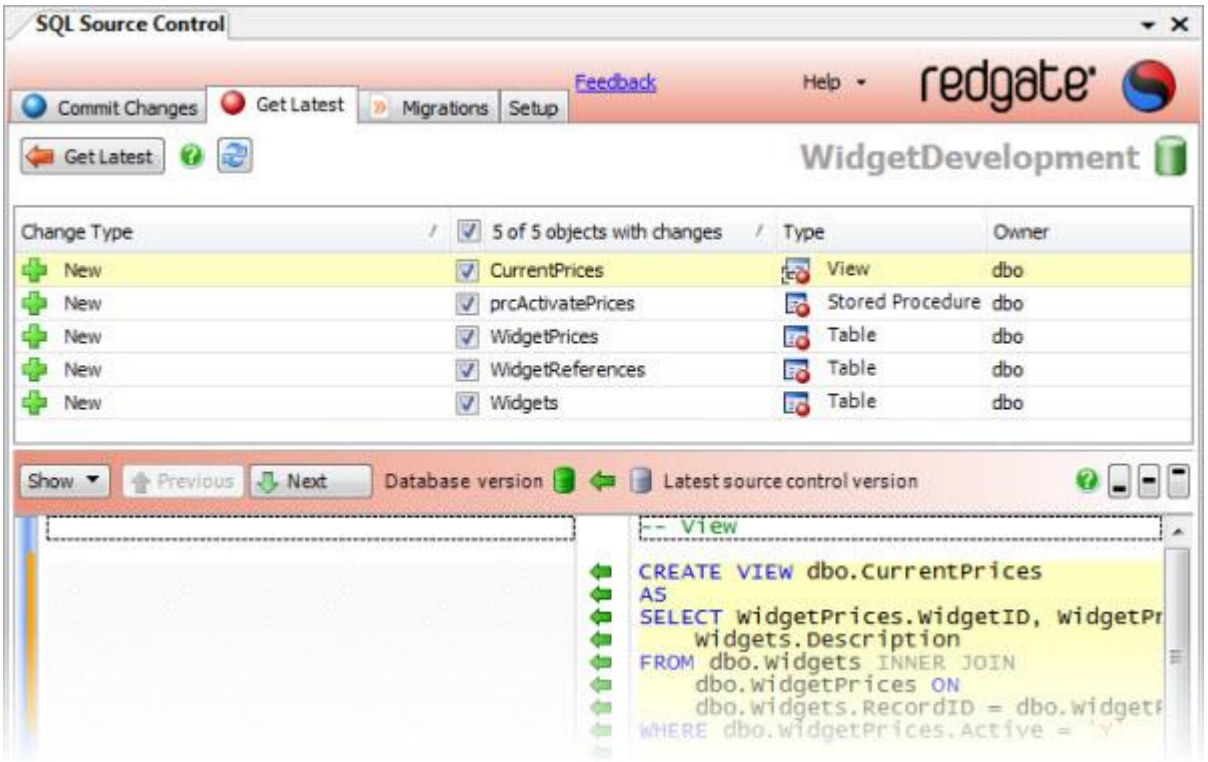
   A link to the repository is created, and SQL Source Control is able to determine differences between the database and the repository.

   However, the database has not yet been updated with the objects from source control.

Getting the latest version

1. Click the **Get Latest** tab.

   Because you do not yet have any of the objects in your database, all the objects in *WidgetDevelopment* are listed here.



2. Ensure all of the objects are selected.
3. Click **Get Latest**.

   A progress dialog box is displayed while SQL Source Control updates the database.

   The database is updated to the latest version.

5. Make development changes

Development proceeds normally, and the database is modified.

A column is changed in the table *Widgets*, to allow longer widget descriptions. This change is committed to source control.

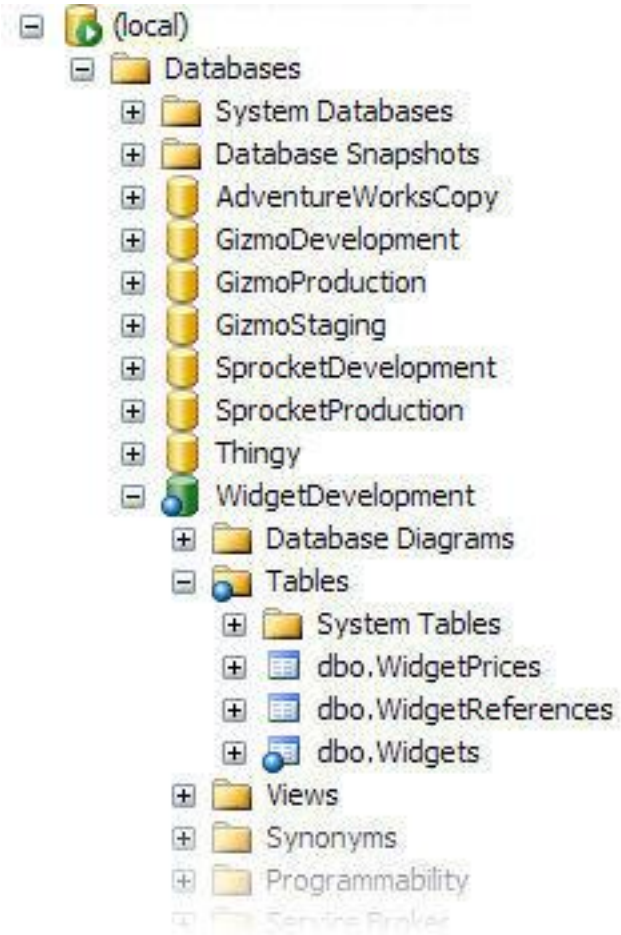
1. In SQL Server Management Studio, open a new query window, and type or paste the following SQL statement:

```
USE WidgetDevelopment
GO
ALTER TABLE [dbo].[Widgets] ALTER COLUMN [Description] [nvarchar] (100)
```

2. Click **Execute** or press **F5**

   The script runs; the database is updated.

3. SQL Source Control detects the change to the database, and highlights the affected object in the Object Explorer:

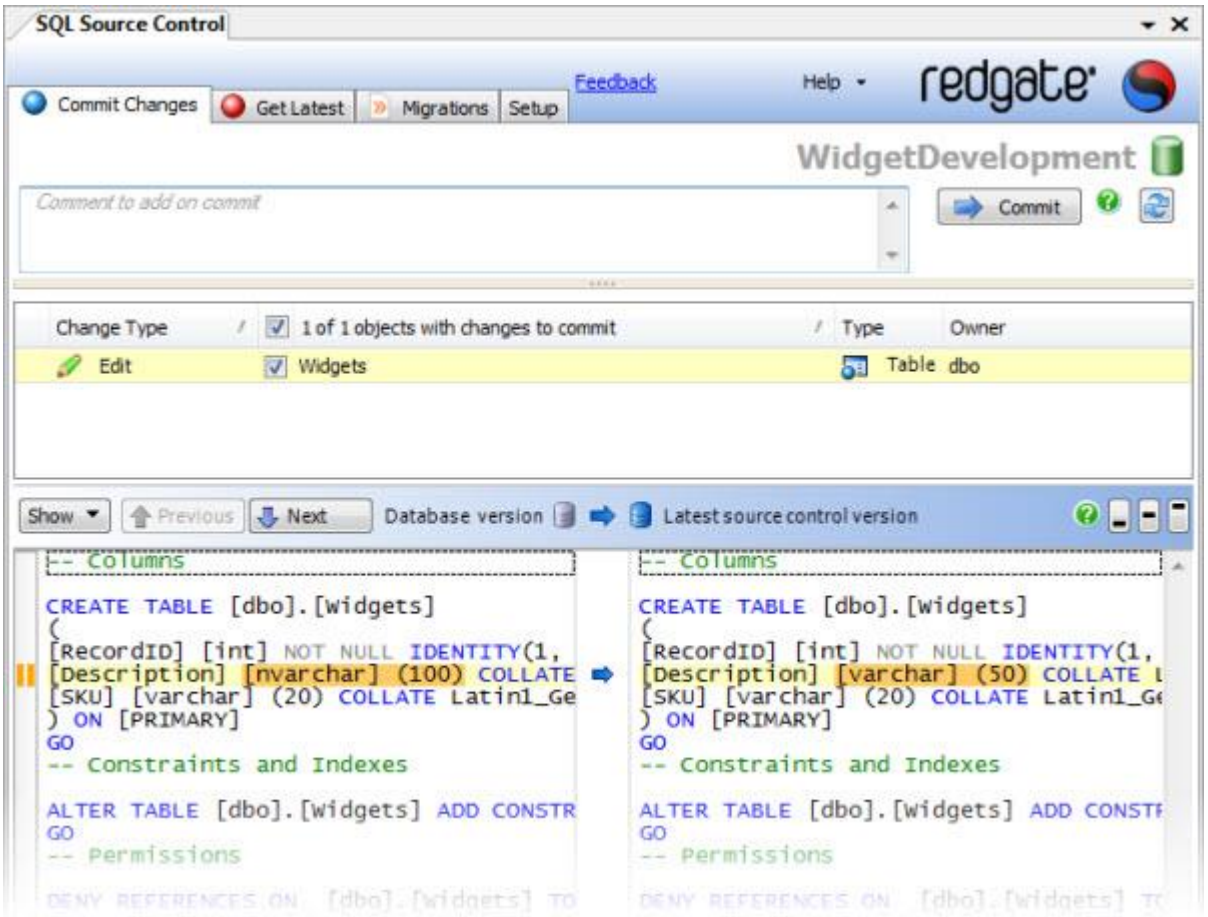


You can now commit the change.

6. Commit the development changes to source control

1. In the Object Explorer, right click the table *Widgets,* and click **Commit changes to source control**

The Commit changes tab is displayed:



The change to the table *Widgets* is listed as an edit you can commit.

The Object Differences pane shows the difference between your database version, and the latest version in source control.

2. In **Comment to add on commit**, type or paste a comment.

   In this example, type *Modified Description column*

3. Click **Commit**

   SQL Source Control displays a message dialog box that shows the progress of the commit.

   When the commit is complete, click **OK** to close the message box.

Your change is committed to source control. The Commit Changes tab lists no objects with changes to commit, and no objects are highlighted in the Object Explorer.

Other users can now get the latest version of the table.

Linking a database to source control

Before you can commit changes to a database, you need to link it to SQL Source Control.

Linking associates a database with a location in source control, allowing SQL Source Control to monitor changes, and manage migration scripts.

When you link, you tell SQL Source Control if a database is shared, so it can alert you to any conflicts or issues with the shared database development model.

For more information, see Database development models (http://www.red-gate.com/supportcenter/Content.aspx?p=SQL%20Source%20Control&c=SQL_Source_Control\articles\SSC_Development_Models.htm).

When linking a database, you have two options:

- **Link the database to source control**

  This is used to source control a database for the first time; to associate your version of a database with one that is already source controlled; or to get a new database from source control.

- **Use the evaluation repository**

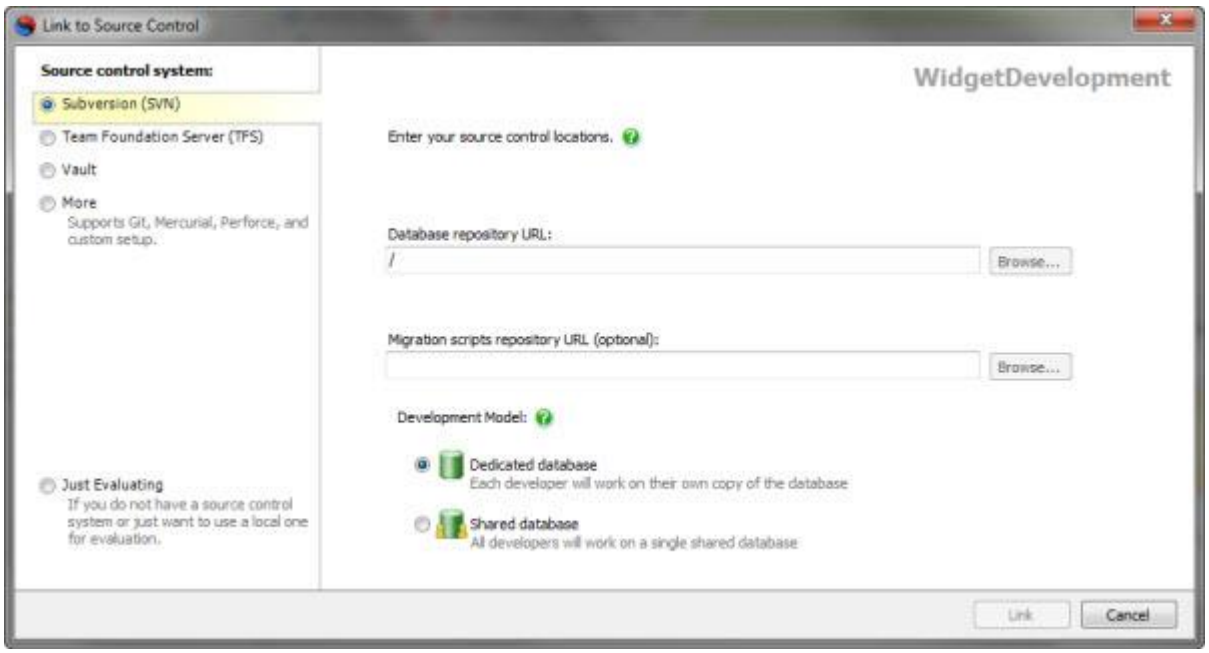
  This creates a local Subversion repository on your computer, so you can evaluate SQL Source Control even if you do not have source control set up.

  For more information, see Using the evaluation repository (page 67).

To link a database to source control:

1. In SQL Source Control, on the **Setup** tab, ensure a database is selected, and click **Link to source control**.

   The Link Database to Source Control dialog box is displayed:



2. On the left hand side, select your source control system; either *Subversion* (SVN), *Team Foundation Server* (TFS), *Vault*, *More* (for custom setup, including Perforce, Git, and Mercurial) or *Just Evaluating*.
3. Provide details for the source control location you want to link the database to.

   For Subversion:

   Type or paste the URL for your Subversion repository.Type or browse to the location of the folder you are linking to.

   For Team Foundation Server:

   Type or paste the URL of your Team Foundation Server and the port number to connect to that server; type or browse to the location of the Team Project and solution you want to associate with the database.

   For Vault

   Type or paste the URL of your Vault server and your repository name; Type or browse to the location of your database folder.

   For other source control systems

   Type or browse to the location of your working folder, then select the config file you are using.

   For more information, see Working with config files (page 52).

   If you are creating a new link you can create a new folder by clicking **Browse**, choosing a location, and clicking **Create Folder**
4. Tell SQL Source Control if the database is shared.

   If you are linking to a database that will be used by multiple developers, ensure the *This is a shared database* radio button is selected.

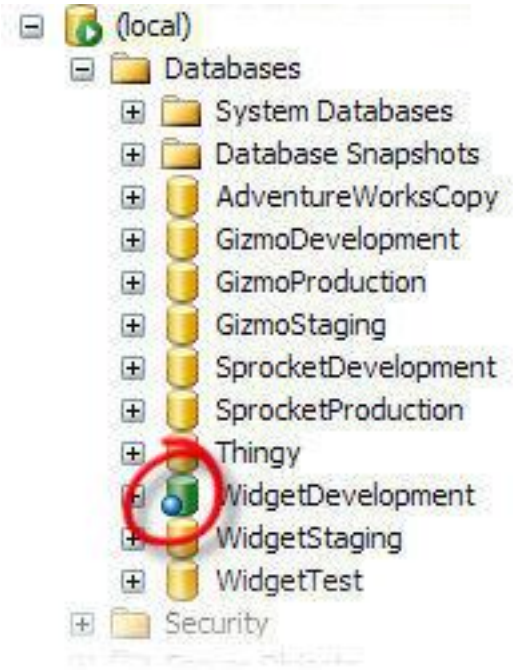
   For more information, see Database development models (http://www.red-gate.com/supportcenter/Content.aspx?p=SQL%20Source%20Control&c=SQL_Source_Control\articles\SSC_Development_Models.htm).
5. Specify a location for your migration scripts repository.

   Migration scripts are customizable change scripts, re-used in deployment. They are stored in source control in a separate folder to your database schema.

   For more information on migration scripts, see Working with migration scripts.
6. Click **Link**

   The database is linked to source control.

The database icon in the Object Explorer changes to show that the database is linked:



Note that linking only associates the database with a location in source control.

If you are creating a new link, the database objects are not yet source controlled.

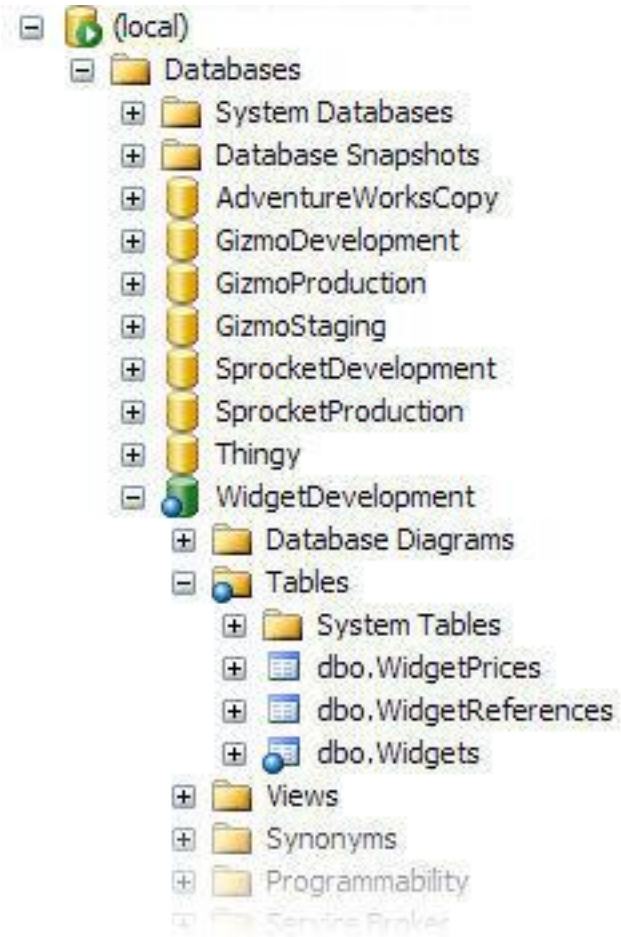
If you are linking to an existing version, your database has not yet been updated to that version.

To commit the objects or get the latest version, next go to either the **Commit Changes** tab or the **Get Latest** tab.

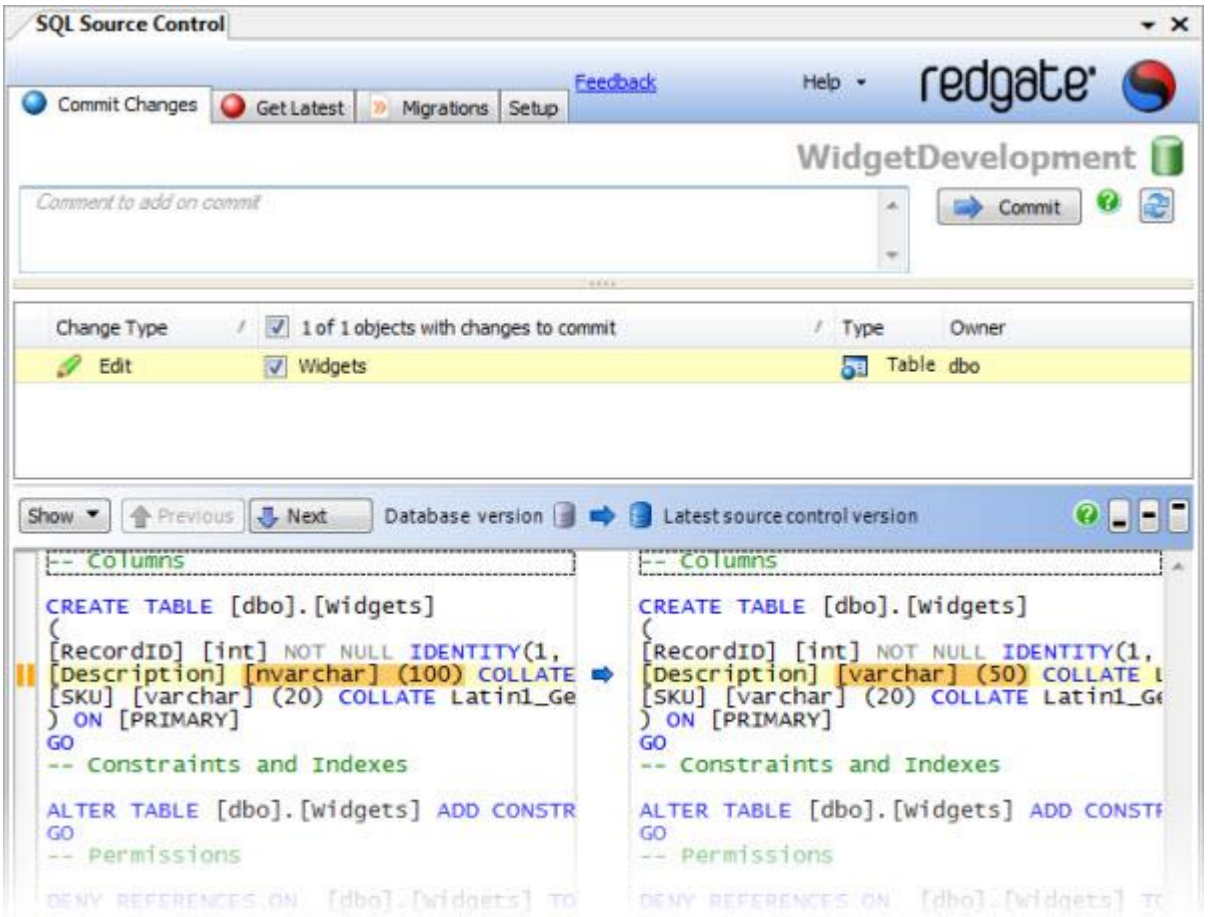
Committing changes

Committing a change updates source control with that change.

When you make changes to a database that is linked to source control, SQL Source Control highlights affected objects in the Object Explorer:



To update source control with your changes, click the **Commit Changes** tab:



The Commit Changes tab lists all the objects in your database that anyone has made changes to.

To commit changes:

1. Select the objects you want to commit.
2. Optionally type or paste a comment.
3. Click **Commit**

   A progress dialog box is displayed while SQL Source Control commits the changes to your source control system.

   Click **OK** to close the dialog box.

Source control is updated with your changes.

You can also commit changes by right-clicking an object, folder, or database in the Object Explorer, and clicking **Commit changes to source control**. The **Commit Changes** tab is displayed. The objects you clicked selected to commit.

Integrating with bug tracking systems

SQL Source Control enables you to integrate your database changes with a bug or issue tracking system (for example Jira, or FogBugz), or the project management features of Team Foundation Server. It does this by letting you associate a commit with an SVN bug ID or TFS work item.

For more information, see: Using SVN Bug IDs & TFS Work Items (page 51)

Source controlling data

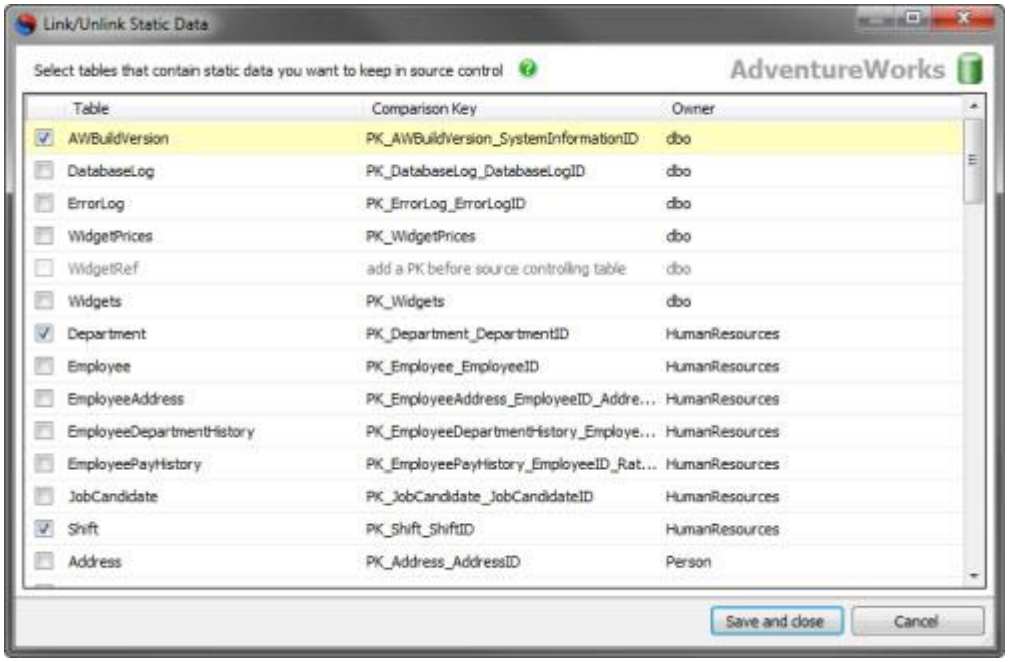
SQL Source Control lets you source control your static (lookup/reference) data. Static data is the non-transactional data an application may depend on. It typically does not change frequently; an example would be a table of US states.

To source control data, first link it to source control, then commit it.

Linking associates a table's data with source control, and means that subsequent data changes are detected, and static data can be shared across the team.

To source control a table's data:

1. In the Object Explorer, right-click the database or table with data you want to source control.
2. Select **Other SQL Source Control Tasks**, then click **Link/Unlink Static Data**
3. The Link/Unlink Static Data dialog box is displayed:



   You can see a list of the tables in the database and select the ones you want to link to source control.

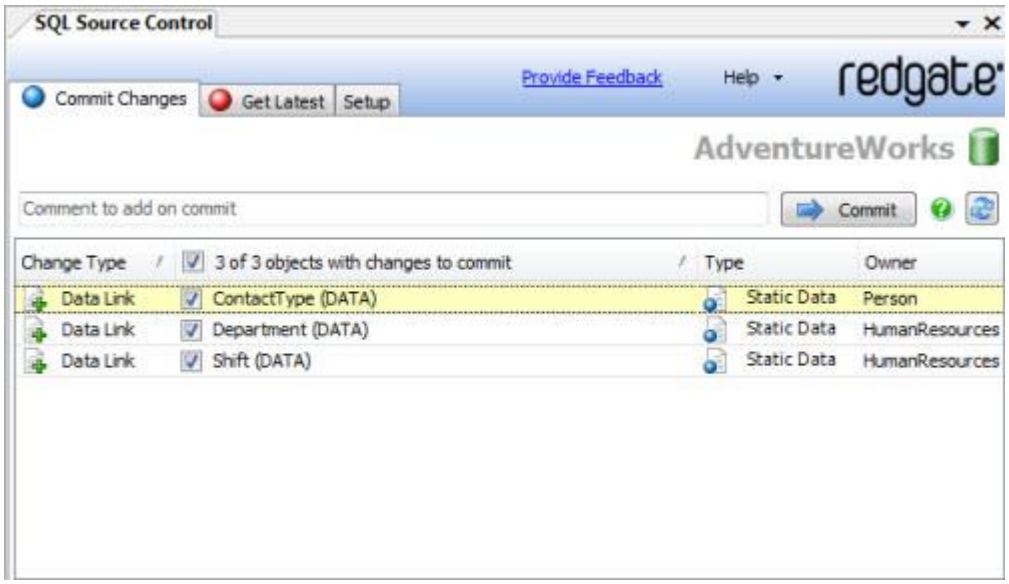
   Note that you can only source control data in tables with a valid primary key.

   The primary key is used as the comparison key (/supportcenter/Content?p=SQL Data Compare&c=SQL_Data_Compare/help/9.0/9067.htm&toc=SQL_Data_Compare/help/9.0/toc1419819.htm) to identify corresponding rows.

4. Select the tables you want to link and click **Save and close**

   A file for the table's data has now been created, but the data itself has not yet been committed. To commit the data, commit the resulting *Data Link* change on the Commit Changes tab.

5. On the Commit Changes tab, ensure all Data Link changes are selected, optionally type a comment, and click **Commit**

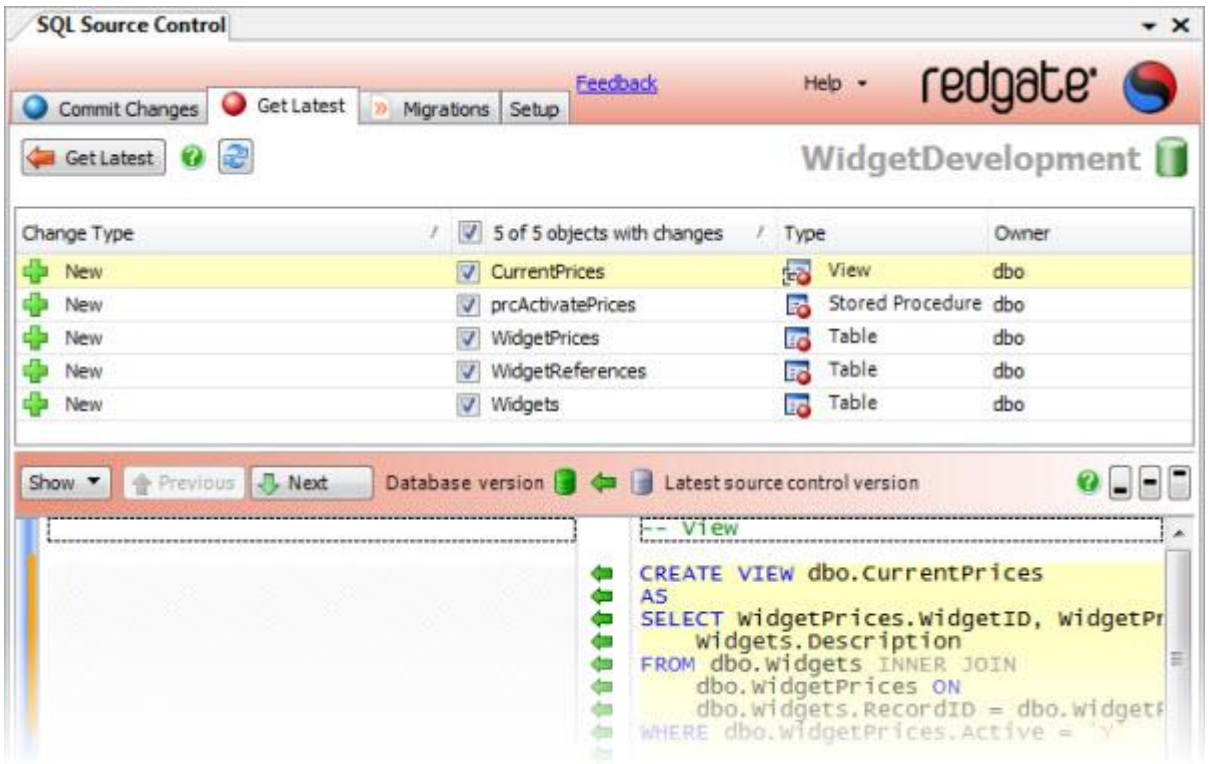


The data is committed to source control.

Note that committing and getting the latest version of data can be slow. It is therefore recommended only to source control static (lookup) data.

Getting the latest version

If an object has a more recent source control version than the version currently in the database, you can get that version on the **Get Latest** tab:



To get the latest version of an object:

1. On the **Get Latest** tab, select the objects you want to update to the latest version
2. Click **Get Latest**

   A progress dialog box is displayed while SQL Source Control updates the database.

The database is updated to the latest version.

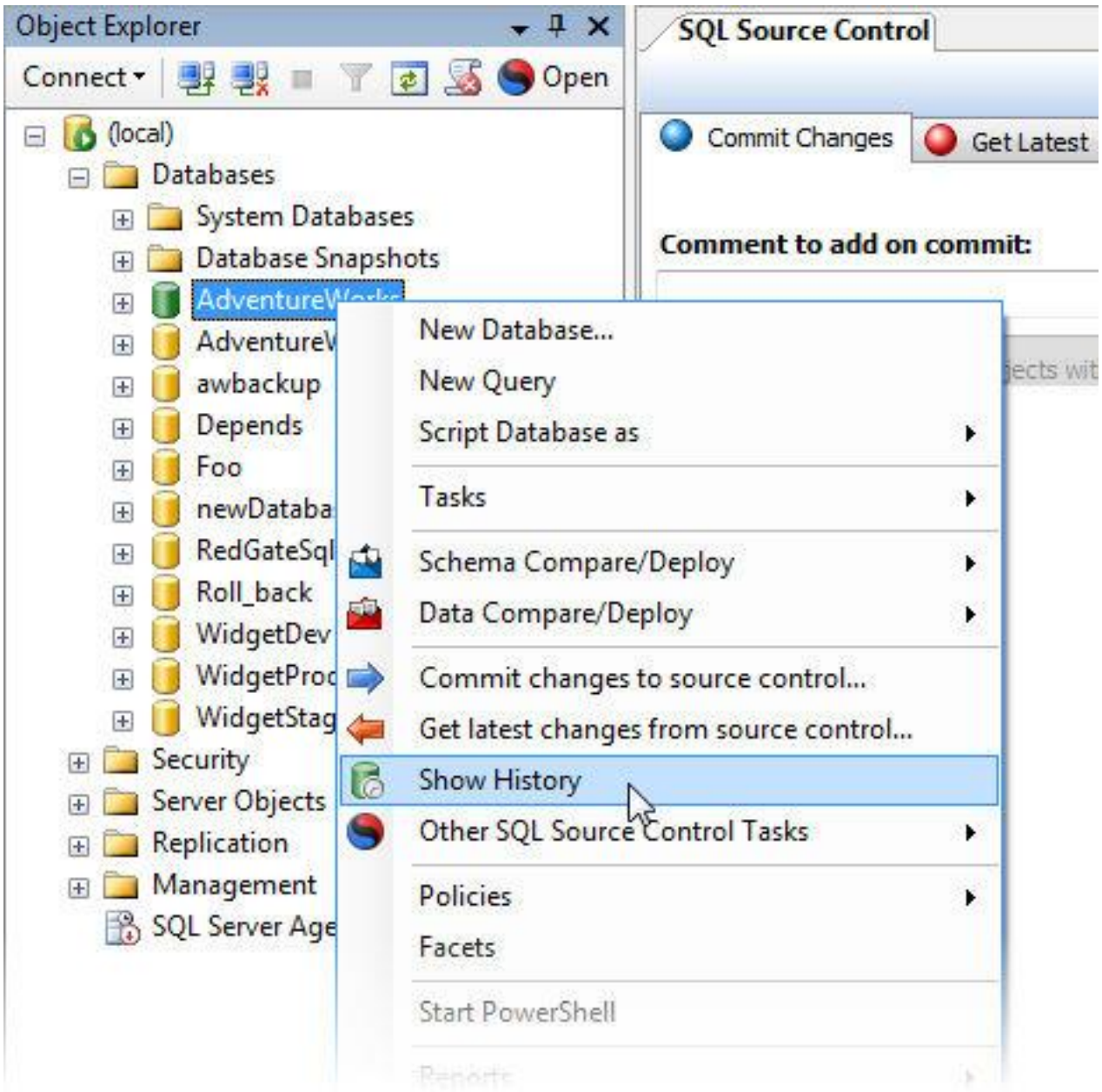
You can also get the latest version by right-clicking an object, folder, or database in the Object Explorer, and clicking **Get latest changes from source control**. You are taken to the **Get Latest** tab, where the objects you clicked are selected.

Note that if you are using a shared database, you never need to get the latest version. This is because the shared database is always up to date with everyone's changes.

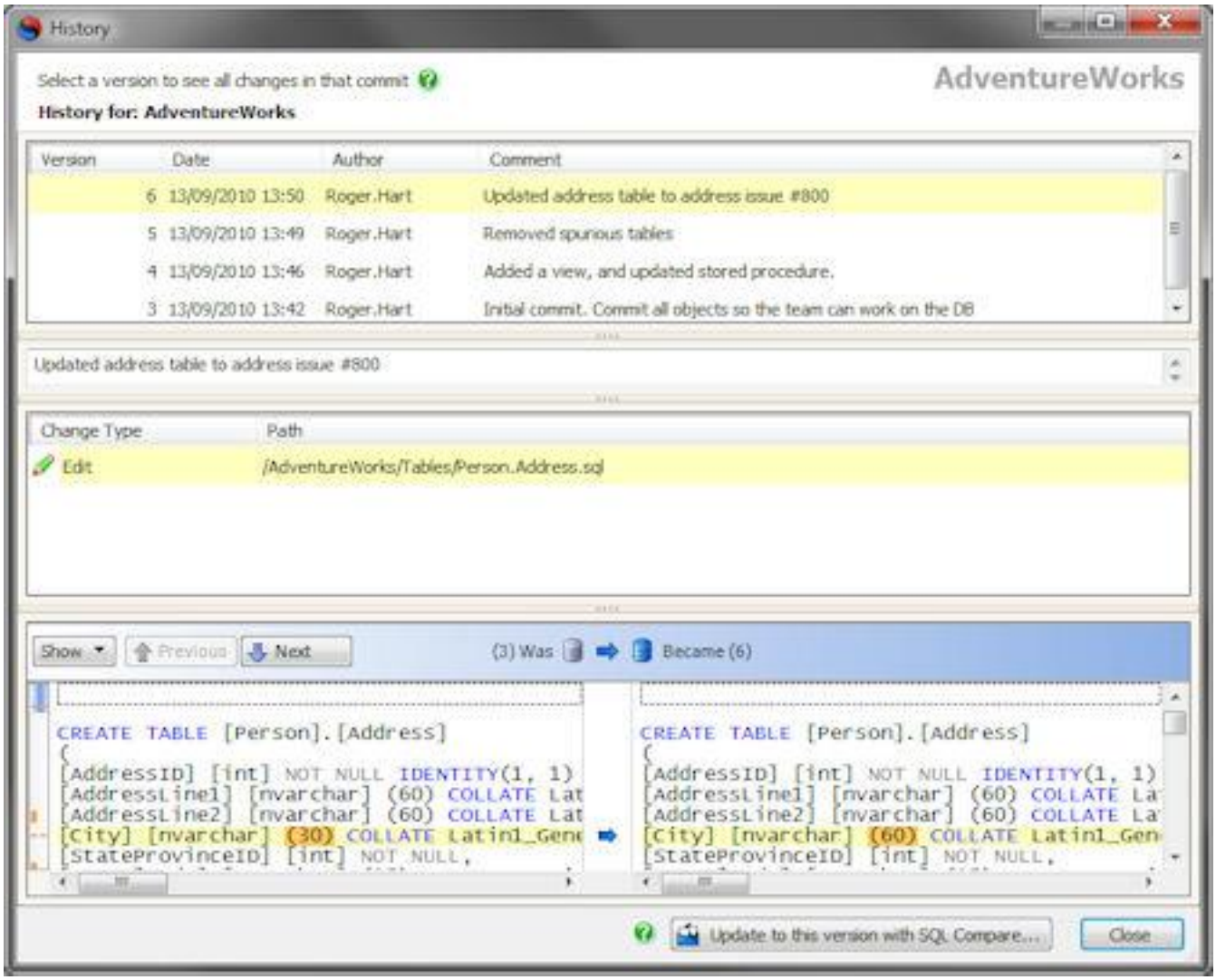
Viewing source control logs / history

You can use SQL Source Control to view the log or history information for a database or database object from within SQL Server Management Studio.

To view the logs / history, in SQL Server Management Studio, in the Object Explorer, right-click the database or object and click **Show History:**



The History dialog box is displayed:



The History dialog box shows:

- Each version; this is the SVN revision, or TFS changeset
- The author, date, and comment associated with each commit
- Which objects changed in each commit
- The SQL differences for each object

You can use the History dialog box to get a specific version of the database (page 42)

Note that you cannot view history information for databases linked to source control using command line support.

Getting a specific version

You can update your database to a specific version using the History dialog box (page 40)

Note that when you do this, the current database schema (and any data you have source controlled) is overwritten with the version you selected.

Getting a specific version within SQL Source Control requires SQL Server Management Studio Integration Pack and SQL Compare 8.50 or later (http://www.red-gate.com/products/sql-development/sql-compare/)

Alternatively, if you do not have SQL Server Management Studio Integration Pack installed, you can manually create a local copy using your source control system, and synchronize it with the database using SQL Compare (http://www.red-gate.com/products/sql-development/sql-compare/)

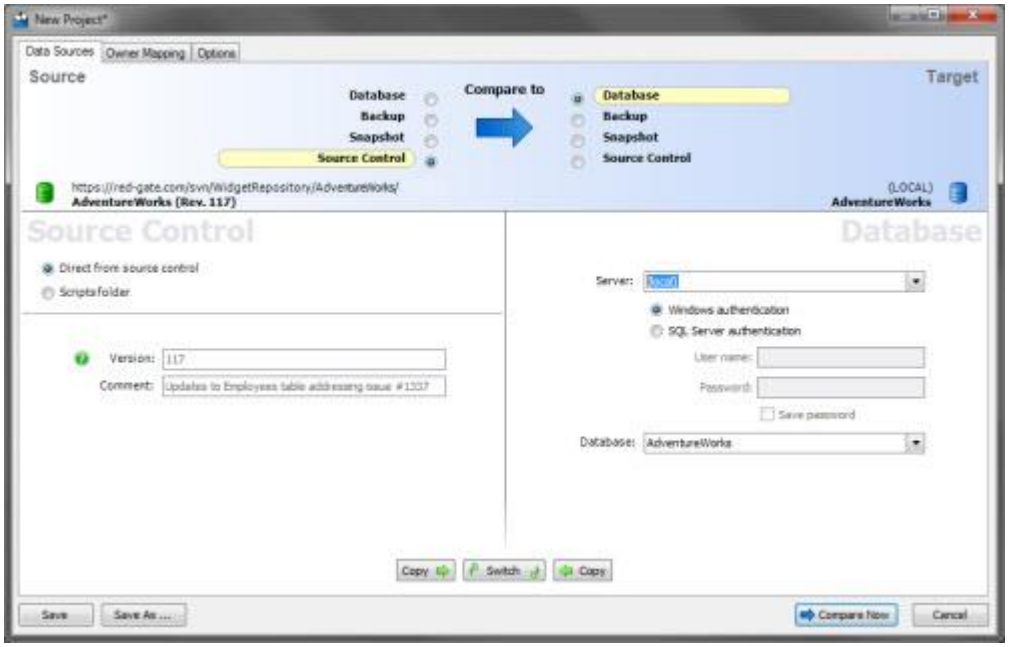
Getting a specific version from the History dialog box

When you get a version form the History dialog box, SQL Compare is used to update the target database.

To get a specific version:

1. On the History dialog box, select the version you want to get.
2. Click **Update to this version with SQL Compare**

   A progress dialog box is displayed while the details of the version you selected are determined.
3. A SQL Compare project launches with the version you selected set as the source, and the database as the target:



4. To update the database, perform the comparison, ensure all objects you want to update are selected, and run the synchronization wizard.

For more information, see Worked example: Comparing and synchronizing two databases http://www.red-gate.com/supportcenter/Content?p=SQL%20Compare&c=SQL_Compare/help/9.0/sc_WorkedExampleCompareAndSync.htm&toc=SQL_Compare/help/9.0/toc141374.htm, in the SQL Compare documentation.

Getting a specific version manually

If you do not have SQL Server Management Studio Integration Pack installed, you can get a specific version using your source control system and SQL Compare.

Example: getting a specific revision with TortoiseSVN

This example creates a local copy of the revision, and synchronizes the database using SQL Compare:

1. Check out the latest version to a new folder.
2. Right-click the folder, and from the **TortoiseSVN** menu, select **Show log**

   The Log Messages dialog box is displayed.
3. Select the revision you want to get, right-click, and click **Revert to this revision**
4. A confirmation dialog box is displayed. Click **Yes**

   The folder of scripts is updated to the revision you selected.
5. Using SQL Compare, set the scripts folder as the source for a comparison, and the database as the target, then compare and synchronize.

   The database is updated to the revision you selected.

Example: getting a specific changeset with TFS

This example updates the TFS local copy to a specific changeset, and synchronizes the database using SQL Compare:

1. In Visual Studio, in the **Source Control Explorer** tab, select the database, right-click, and click **Get Specific Version**

   The **Get** dialog box is displayed.
2. Under **Version**, in **Type** select *Changeset*
3. Type the changeset number, or click the browse button to display the Find Changesets dialog box, and select the changeset you want.
4. Click **Get**

   The local scripts folder has been updated to the changeset you selected.
5. Using SQL Compare, set the local scripts folder as the source for a comparison, and the database as the target, then compare and synchronize.

   The database is updated to the revision you selected.

Deploying a database from source control

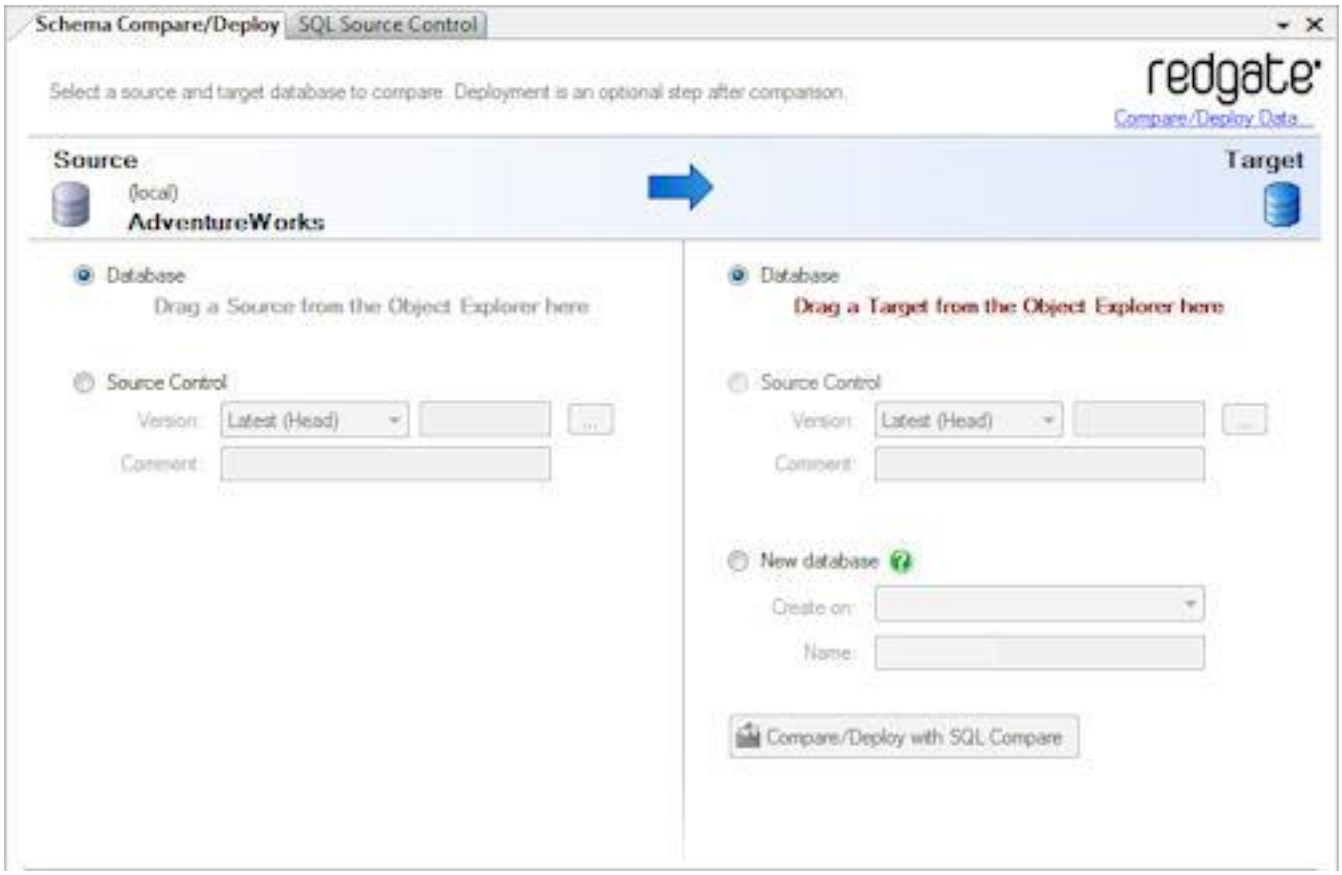
You can use SQL Source Control and SQL Compare (http://www.red-gate.com/products/SQL_Compare/index.htm) to deploy a database from source control to a server.

You can also use the SQL Server Management Studio Integration Pack add-in to make schema and data deployment simpler.

Deploying with SQL Server Management Studio Integration Pack

To deploy a database schema, in the Object Explorer, right-click a database, select **Schema Compare/Deploy**, and click **Set as Source**

The SQL Server Management Studio Integration Pack Schema Compare/Deploy tab is displayed:



You can deploy the current database version or specify a version from source control.

You can deploy to a target database, create a new database, or create a change script to update a target source control version.

For more information, see Getting started with the add-in (http://www.red-gate.com/supportcenter/Content?p=SQL%20Compare&c=SQL_Compare/help/9.0/ssmsip_getting_started.htm&toc=SQL_Compare/help/9.0/toc1413764.htm)

Deploying without SQL Server Management Studio Integration Pack

To deploy a database:

1. Create a local copy of the scripts folder
2. Migrate the local copy to the target server using SQL Compare

Optionally, you can also deploy any relevant static data using SQL Data Compare.

In this example the database *WidgetDev* is already in source control.

The example uses the SQL Compare and Subversion command line interfaces.

It is also possible to deploy the database using the SQL Compare graphical user interface, and a source control client such as TortoiseSVN (http://tortoisesvn.net/downloads)

1) Create a local copy of the database

At a command prompt, type:

```
cd C:\program files\subversion\bin
svn update http://<your repository path>/WidgetDev "C:\WidgetDevScripts"
```

Where:

- *http://<your repository path>/WidgetDev* is the URL for the database in your Subversion repository
- *"C:\WidgetDevScripts"* is the file path for the directory where the local copy will be created

A local copy of the scripts folder is created. This is a Subversion working copy, and is associated with the Subversion repository.

2) Migrate the local copy to the target server

At a command prompt, type:

```
cd C:\program files\red gate\SQL Compare 8
     /sqlcompare /scr1:"C:\WidgetDevScripts"
     /S2:WidgetServer /U2:<username> /P2:<password>
     /db2:"WidgetTest"
     /sync
```

Where:

- */scr1:"C:\WidgetDevScripts"* specifies the local copy, WidgetDevScripts, as the source for a SQL Compare synchronization
- */S2:WidgetServer /U2:<username> /P2:<password>* specify the server, user name, and password you are using
- */db2:WidgetTest* specifies WidgetTest as the target of a SQL Compare synchronization
- */sync* performs the SQL Compare synchronization, making schema of WidgetTest the same as the schema in WidgetDevScripts

The database is updated. Its schema is now the same as the version you checked out of source control.

For more information, see Simple examples using the SQL Compare command line (http://www.red-gate.com/supportcenter/Content.aspx?p=SQL%20Compare&c=SQL_Compare/help/9.0/sc_cl_examplesusingthecl.htm&toc=SQL_Compare/help/9.0/toc846154.htm)

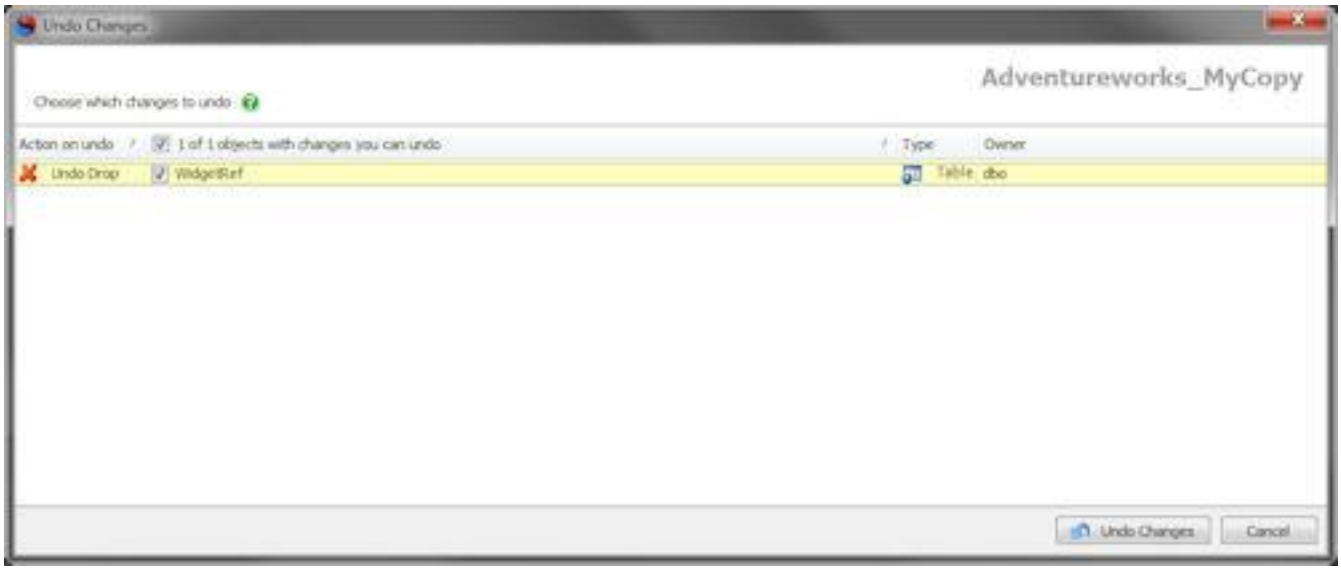
Undoing changes

SQL Source Control enables you to undo changes you have made to a database, but not yet committed to source control.

To undo a change:

1. In the Object Explorer, select the object, folder, or database that has changes you want to undo.
2. Right-click, select **Other SQL Source Control Tasks** and click **Undo changes**

   The Undo Changes dialog box is displayed:



3. Select the objects with changes you want to undo, and click **Undo Changes**

   A progress dialog box is displayed while SQL Source Control runs the script to undo the changes.
4. Close the dialog box when the undo is complete.

   The changes are undone.

Alternatively, right-click an object on the Commit Changes tab, and click **Undo Changes**

Changes you cannot undo

The most common types of change SQL Source Control cannot undo are:

- **Committed changes**

  You can only use SQL Source Control to undo changes you have not committed. To undo changes that have been committed, use SQL Server Management Studio Integration Pack to update the database to a specific version (page 42) or use your source control system.

- **Static data changes**

  You cannot currently undo a data link or a data edit. To stop source controlling a table's data, right-click it in the object explorer, and click **Link/Unlink Static Data**

  On the Link/Unlink Static Data dialog box, you can choose to stop source controlling the table's data.

- **Dropped data**

  If you drop a table or column that contained data, the data is not restored when you undo the drop.

- **NOT NULL columns**

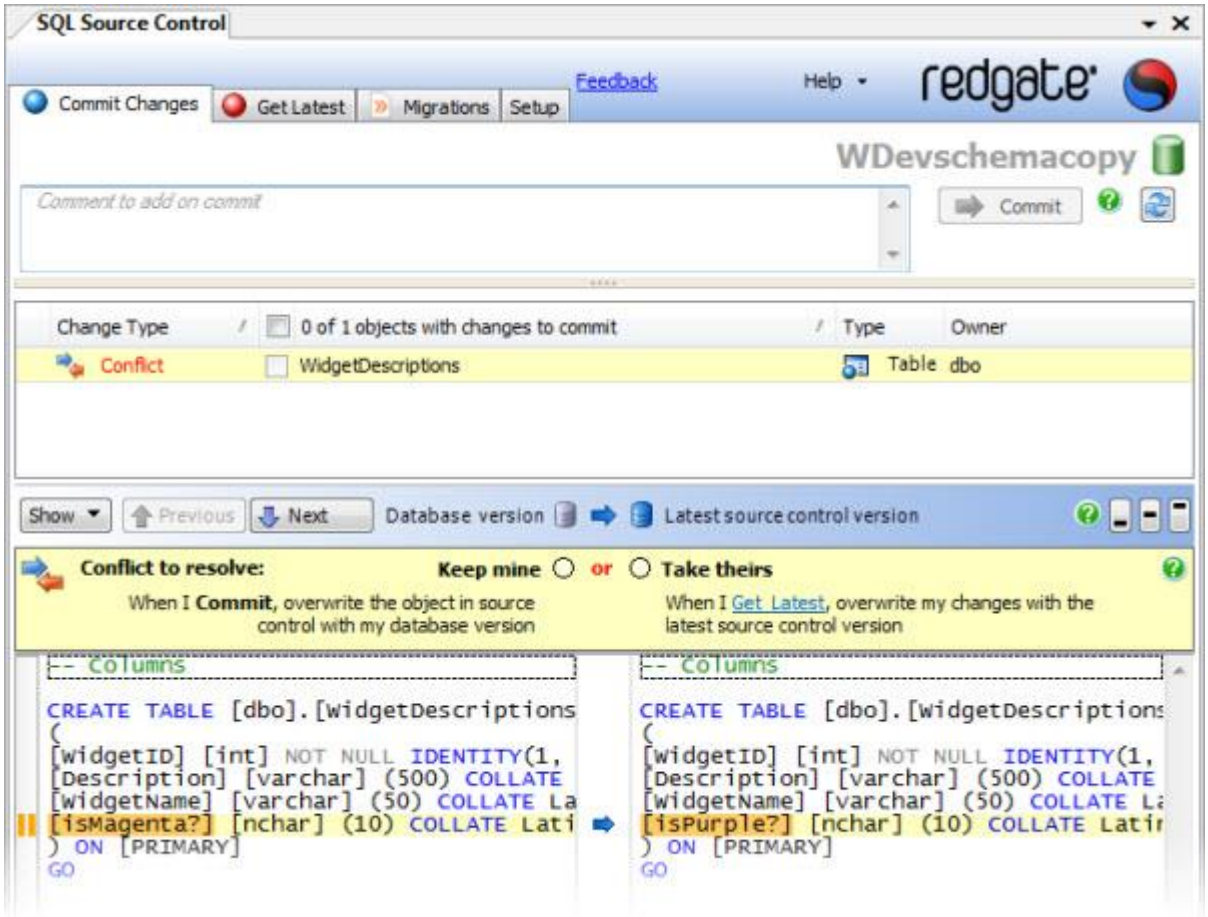
  If you drop a NOT NULL column from a table that contained data, undoing the drop will fail if the column does not have a default value.

Conflicts

Conflicts occur when the latest version of an object in source control and the latest version in the database are incompatible. Conflicted objects cannot be committed or retrieved until the conflict has been resolved.

This happens if two people modify the same object.

When a conflict occurs, the **Conflict to resolve** bar is displayed in the Object Differences pane:



To resolve a conflict, choose either:

- **Keep mine**

  Your changes are committed to source control.

  This replaces any changes to the latest source control version that are not present in your database version.

  Or

- **Take theirs**

  Your changes are discarded, and the database is updated to the latest source control version.

Merging

Note that merging is not currently supported.

To merge changes from two conflicting versions of an object, resolve the conflict, and then manually edit the object to include the changes from the other version.

To make this easier, you can copy either version of an object's creation script from the Object Differences pane, and paste it into a new query window.

To avoid conflicts, you are recommended always to get the latest version of an object before modifying it.

Conflicts involving data

If an object has both schema and data changes, you must commit or get the latest schema and data changes at the same time. You cannot commit only the data changes. This is because data changes may fail without the associated schema changes.

So if an unconflicted data change is associated with a conflicted object, you cannot commit or get the data change without first resolving the schema conflict.

Using SVN bug IDs & TFS work items

You can use SQL Source Control to associate your changes with an SVN bug ID or a TFS work item.

This allows you to better integrate database changes with a bug or issue tracking system, or the project management functionality of Team Foundation Server.

Currently, SQL Source Control only supports this by adding the bug ID or work item number to the comment you make when committing changes.

SVN Bug IDs

SVN bug IDs are enabled using *bugtraq* properties. These let you add commit hooks to make SVN parse log messages for bug or issue numbers, and relate them to your issue tracking system.

For more information on setting up SVN Bug IDs, see TortoiseSVN: Integration with an issue tracker (http://tortoisesvn.net/issuetracker_integration)

To associate a commit with a bug or issue, include the issue number in the commit comment, with a **#** symbol.

For example: *This commit addresses issue #100*

TFS work items

TFS work items are a way of tracking pieces of work in a development project. Each work item has a number, and this number can be included in your commit comment to either associate the commit with a work item, or to mark it as resolving a work item.

To associate a commit with a work item include *#**A**[Work Item number]* in the Comment.

For example: *#A106*

To resolve a work item include *#R[Work Item number]* in the Comment.

For example: *#R106*

Note that the *#A106* and *#R106* portion will not appear in the Commit Comment recorded on the TFS Server.

Working with config files

SQL Source Control provides preset XML config files that enable you to automate source control operations for source control systems with a command line interface.

Commands in the config files

The preset config files include the following commands:

Get Latest

Updates the local working folder with latest version in source control.

Commit

Commits all changes in the local working folder to source control.

Add

Adds new files to the local working copy. Changes can then be committed to source control using the Commit command.

Edit

Makes the local working copy of the file(s) available for editing. Changes can then be committed to source control using the Commit command.

Delete

Deletes the file(s) from the local working copy. Changes can then be committed to source control using the Commit command.

Revert

Undoes changes if an error occurs during a commit.

Creating a custom config file

To create a config file for your source control system:

1. On the Link to Source Control dialog box, select **More** as your source control system.
2. Click **Manage Config Files**.
3. Open *Template.xml* in your XML editor, and specify commands and verification codes for the source control actions you want to automate. Help is provided inside *Template.xml.*

Note: Make sure you specify the name you want to use for the config file in the <Name> tag. For example:

```
<Name>Darcs</Name>
```

To include multiple operations in a single command, separate them with &&. For example:

```
accurev add -c -d ($Files) && accurev keep -m && accurev promote -k
```

1. Save your changes as a new XML file in the default config files folder:

   %USERPROFILE%\AppData\Local\Red Gate\SQL Source Control 3\CommandLineHooks

   Your config file is now available to select in SQL Source Control.

Using filters to exclude objects

You can create filters to exclude objects from SQL Source Control.

When you exclude an object with a filter, it is not shown on the Commit Changes tab, the Get Latest tab, or the Undo Changes dialog box. This means you can never commit, get, or undo an excluded object.

This is useful, for example, if there is a set of objects that you never want to commit, or consider for source control purposes.

Creating and editing filters

There are two ways to create or edit your filter:

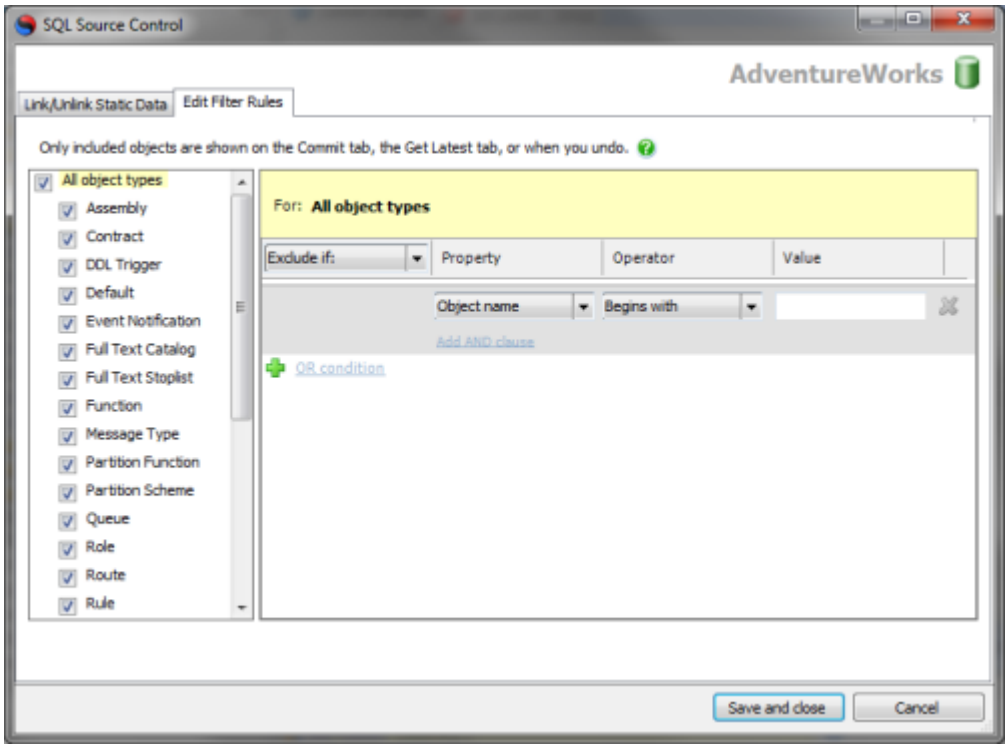
- In the Object Explorer right-click a database, folder, or object, select **Other SQL Source Control Tasks**, and click **Edit Filter Rules...**

  The Edit Filter Rules dialog box is displayed.

- On the Commit Changes or Get Latest tab, right-click an object in the list of changes, and click **Edit Filter Rules...**

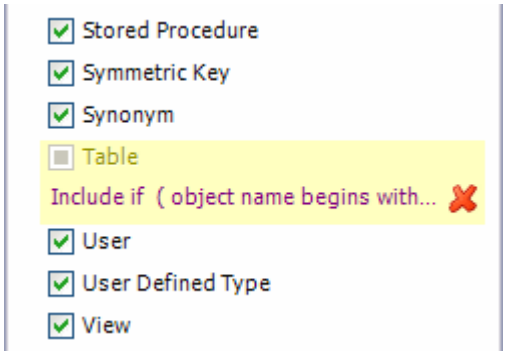
  The Edit Filter Rules dialog box is displayed.

The Edit Filter Rules dialog box lets you specify exclusion or inclusion conditions for individual objects or all object types:



You can exclude object types using the check boxes in the left-hand pane, or build more complex conditions by specifying AND clauses and OR conditions in the right-hand pane.

When you create a filter rule, its conditions are displayed in the left-hand pane, under the name of the object type it applies to:



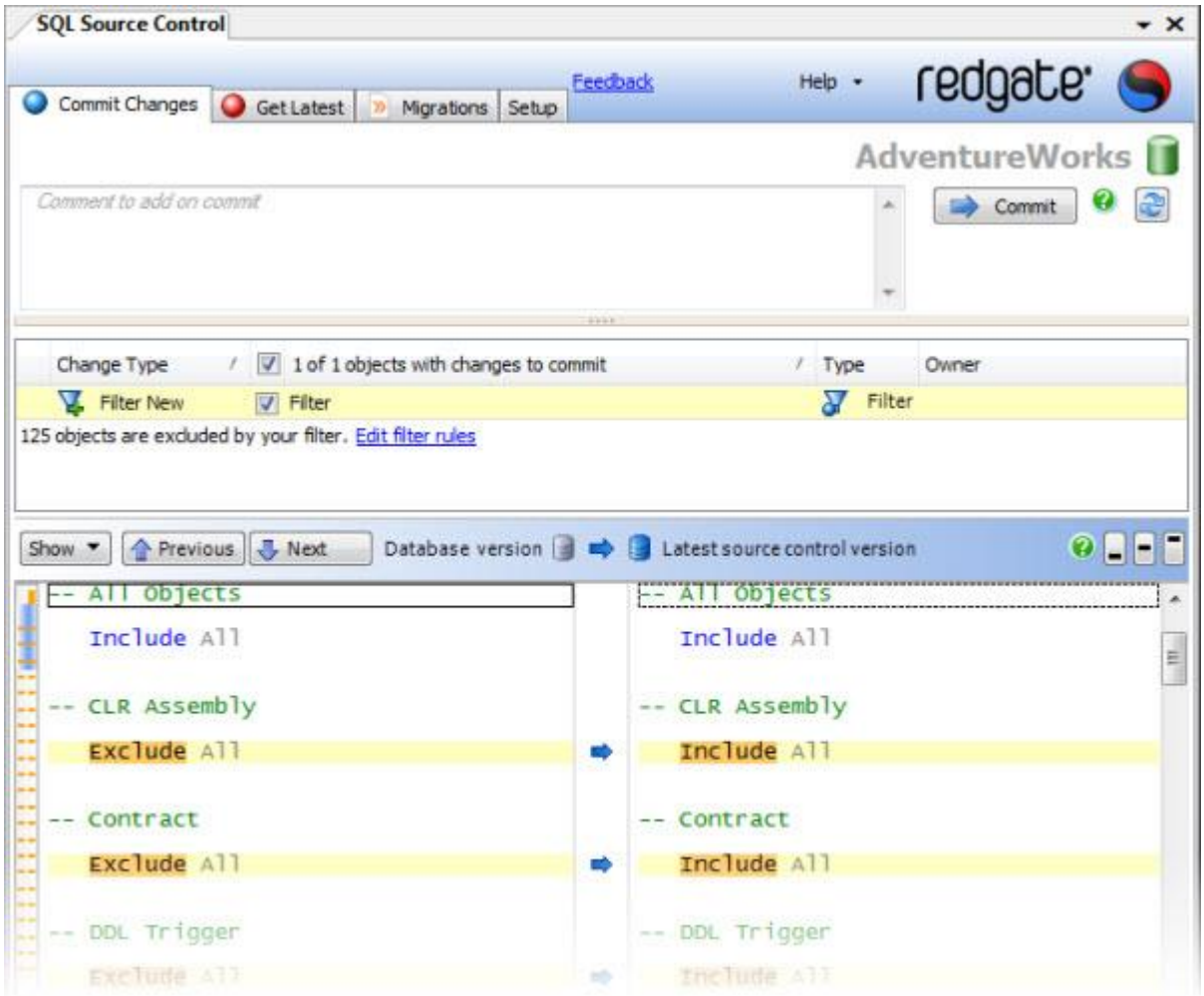
To clear the filter rule for an object type click the (Clear) button next to its name.

Sharing filters

You can commit your filter to source control, and get the latest version when it changes.

This is useful for teams that want to exclude a set of objects across an entire development project.

When you create or edit a filter, SQL Source Control shows it on the Commit Changes tab:



You can see the differences between your current filter and the version in source control.

Example: excluding objects with a specific name or owner

If you never want to commit changes to any tables that have names beginning with *Marketing*, or any owned by the schema *Marketing*, regardless of their names:

1. In the Object Explorer, right-click the database, select **Other SQL Source Control Tasks**, and click **Edit Filter Rules...**

   The Edit Filter Rules dialog box is displayed.

2. In the Exclude if: box, ensure *Exclude if* is selected
3. Under **Property**, select *Object name*.
4. Under **Operator**, select *Begins with*.
5. Under **Value**, type *Marketing*.

6. Click OR condition

   A new OR condition becomes available.
7. Under **Property**, select *Schema name*.
8. Under **Operator**, select *Equals*.
9. Under **Value**, type *Marketing*.
10. Click **Save and close**.

The filter is applied. All objects owned by the schema *Marketing,* or with names that begin with *Marketing* are excluded by SQL Source Control.

You can commit the filter if you want to share it with the rest of the development team.

Branching and merging

Many development projects involve creating branches, for a feature, a release, or some other development milestone. In some source control systems, branches are referred to as "forks". A branch is essentially a copy of the code base that shares its history. Branches diverge as required, and the process of reincorporating the changes from a branch is referred to as merging.

Development projects typically have a "trunk" which is the main code base, and branches which diverge from it.

For more information see:

- Branching and Merging with Team Foundation Server (http://msdn.microsoft.com/en-us/library/ms181423%28v=vs.80%29.aspx)
- Branching and Merging with Subversion (http://svnbook.red-bean.com/en/1.0/ch04.html)

For a general introduction to source control concepts, see:

- Version Control by Example (http://www.ericsink.com/vcbe/) by SourceGear founder Eric Sink.

SQL Source Control allows you to work with branches of a database, although it does not currently provide the means to create a branch or merge branches in SQL Server Management Studio.

To branch with SQL Source Control you must create the branch using your source control system, and then link to the appropriate branch.

For detailed information on branching, refer to the documentation for your source control system.

Working with branches

Once you have created a branch in source control, you are ready to work on it in SQL Server Management Studio with SQL Source Control.

There are two approaches to working with branches:

Un-link and re-link to the branch

Here, you continue working on the same database in SQL Server Management Studio, but link it to the branch in your source control system.

Once the branch is created, un-link it in SQL Source Control, then link the database to source control again. When you link, specify the location of the branch in source control.

Create a new database for the branch

Here, you create the branch in your source control system, and create a new database to link with it in SQL Server Management Studio.

Create a new empty database, and link it to source control in SQL Source Control. When you link, specify the location of the branch in source control, then on the Get Latest tab, update the database with the latest version from source control.

Merging

SQL Source Control does not provide automatic or line-by-line merging functionality. You can use SQL Source Control or SQL Compare (/supportcenter/Content?p=SQL%20Compare&c=SQL_Compare/help/9.0/SC_Getting_Started.htm&toc=SQL_Compare/help/9.0/toc.htm) to merge at an object level, but not to choose line-by-line changes.

When you merge with SQL Source Control or SQL Compare, you choose a version of each object to keep; for example you might keep the trunk version of a table, and the branch version of a view.

There are three approaches to merging:

Merging with your source control system

You can manually merge the branch changes back into the trunk using your source control system, as you would for application code.

This approach is recommended if the merge is complex, or if there are conflicts. For example if the same object has been modified in both the branch and the trunk.

When merging manually, you must ensure that referential integrity is maintained. Otherwise the database could be left in an invalid state.

Your source control system may include auto-merging functionality that simplifies manual line-by-line merges.

Merging with SQL Source Control

If you do not need to perform a line-by-line merge, you can merge with SQL Source Control.

To merge branch changes back into the trunk with SQL Source Control:

1. Ensure that in SQL Source Control you have a database linked to the branch in your source control repository.
2. Get the latest version and commit any outstanding changes.
3. Unlink the database from the branch.
4. Re-link the database to the trunk.
5. Go to the **Commit Changes** tab.

   The tab shows the changes to the branch as changes to commit.

If there are conflicts, choose *Keep mine* to override the trunk with the objects from the branch.

6. Commit the changes.

   The trunk is updated with the branch changes.

Merging with SQL Compare

If there are no conflicting changes between the branch and the trunk, you can merge automatically using SQL Compare.

To merge branch changes into the trunk with SQL Compare:

1. In SQL Source Control, ensure you have a database linked to the trunk.
2. Use your source control system to create a local copy of the latest branch version, for example by performing an SVN checkout.
3. In SQL Compare, create a new project (/supportcenter/Content?p=SQL%20Compare&c=SQL_Compare/help/9.0/sc_working_with_projects.htm&toc=SQL_Compare/help/9.0/toc1413719.htm). Set the local copy of the branch as the *source*, and the trunk database as the *target*.

   For more information, see: Setting data sources (/supportcenter/Content?p=SQL%20Compare&c=SQL_Compare/help/9.0/sc_setting_data_sources.htm&toc=SQL_Compare/help/9.0/toc1413719.htm)
4. Compare the data sources.

   SQL Compare shows the differences between the branch and the trunk.
5. In the Results pane, select the objects from the branch that you want to merge into the trunk, and run the Synchronization Wizard, to synchronize the data sources (/supportcenter/Content?p=SQL%20Compare&c=SQL_Compare/help/9.0/sc_setting_up_synchronization.htm&toc=SQL_Compare/help/9.0/toc1413731.htm)

   The trunk is updated with the changes from the branch.
6. In SQL Source Control, on the **Commit Changes** tab, commit the trunk changes to source control.

Setting up a local Subversion repository

This topic describes how to set up a local Subversion repository using TortoiseSVN, a free Subversion client for Windows.

Note that you are recommended not to use a local repository. Instead, set up a Subversion server (page 63)

To download the latest version of TortoiseSVN, see the TortoiseSVN download page (http://tortoisesvn.net/downloads)

Alternatively, you can use the Subversion command line interface.

For more information, see the Subversion documentation (http://svnbook.red-bean.com/)

Creating a repository

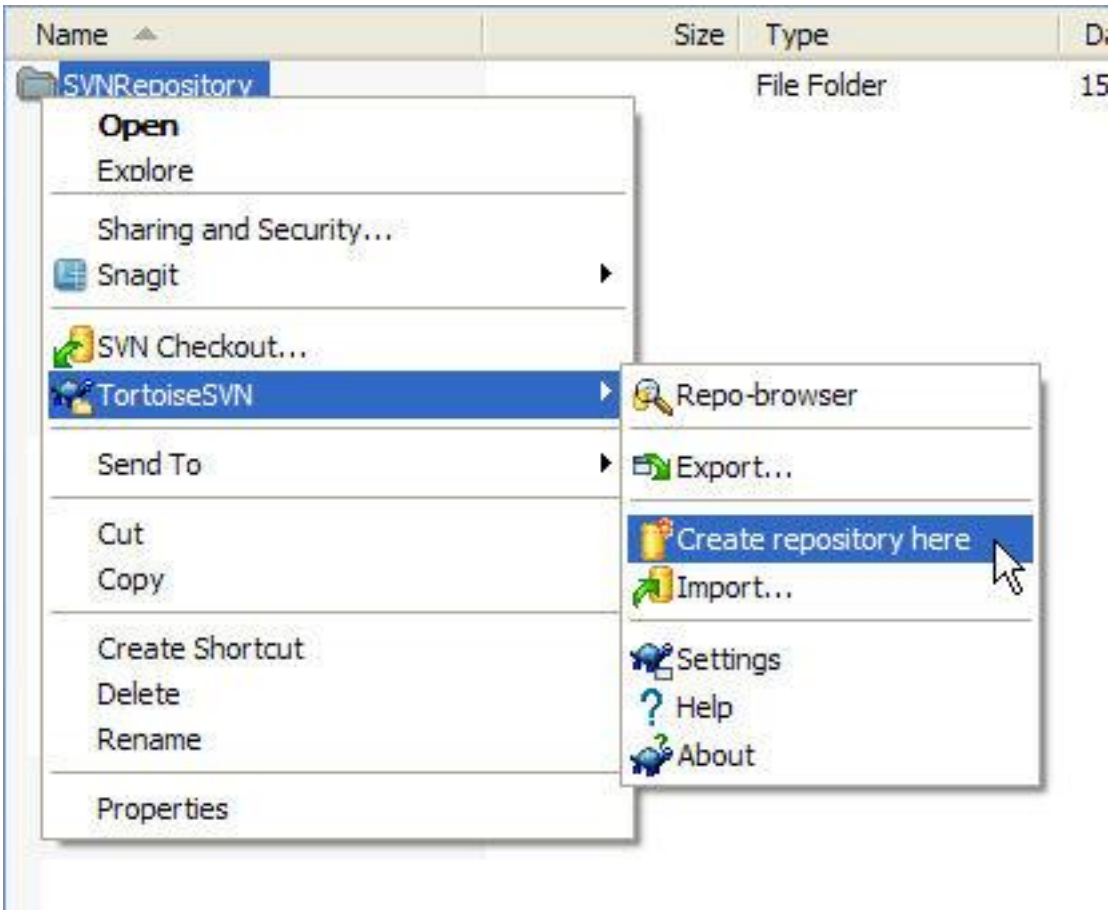
To create a local repository:

1. Download and install Tortoise SVN.

   You may need to restart your computer after installation.

2. In Windows Explorer, browse to or create an empty folder where you want to create the repository, for example *C:\SVNRepository*

3. Right-click the folder, and in the TortoiseSVN menu, select **Create repository here**:



The repository is created.

Using the repository

You can now use the repository with SQL Source Control.

The URL for a local repository takes the form:

file:\\\C:\<RepositoryFilePath>

Use this URL to link your database to source control (page 31)

Note that the URL is case sensitive.

Setting up a Subversion server

This topic provides a simple overview of setting up a Subversion server using VisualSVN Server (http://www.visualsvn.com/server/), an installation and administration application for Subversion on Microsoft Windows servers. This topic does not address manual installation and configuration of Subversion, or installation on non-Windows servers.

For more detailed information on setting up a Subversion server, see:

- Chapter 6 of the Subversion documentation - Server Configuration (http://svnbook.red-bean.com/en/1.0/ch06.html)
- Chapter 3 of the TortoiseSVN documentation - The Repository (http://tortoisesvn.net/docs/release/TortoiseSVN_en/tsvn-repository.html)

Installing with VisualSVN Server

VisualSVN Server automates the setup of a Subversion server, and is available both as a free tool (the Standard Edition), and as the paid Enterprise Edition. The Enterprise Edition includes Integrated Windows Authentication, as well as richer logging and administration tools.

This example uses the free version.

To set up SVN, download and run the VisualSVN Server installer on the server you want to use, then follow the wizard to complete the installation.

You can download the VisualSVN Server installer here (http://www.visualsvn.com/server/download/)

VisualSVN Server provides an installation getting started guide (http://www.visualsvn.com/server/getting-started/)

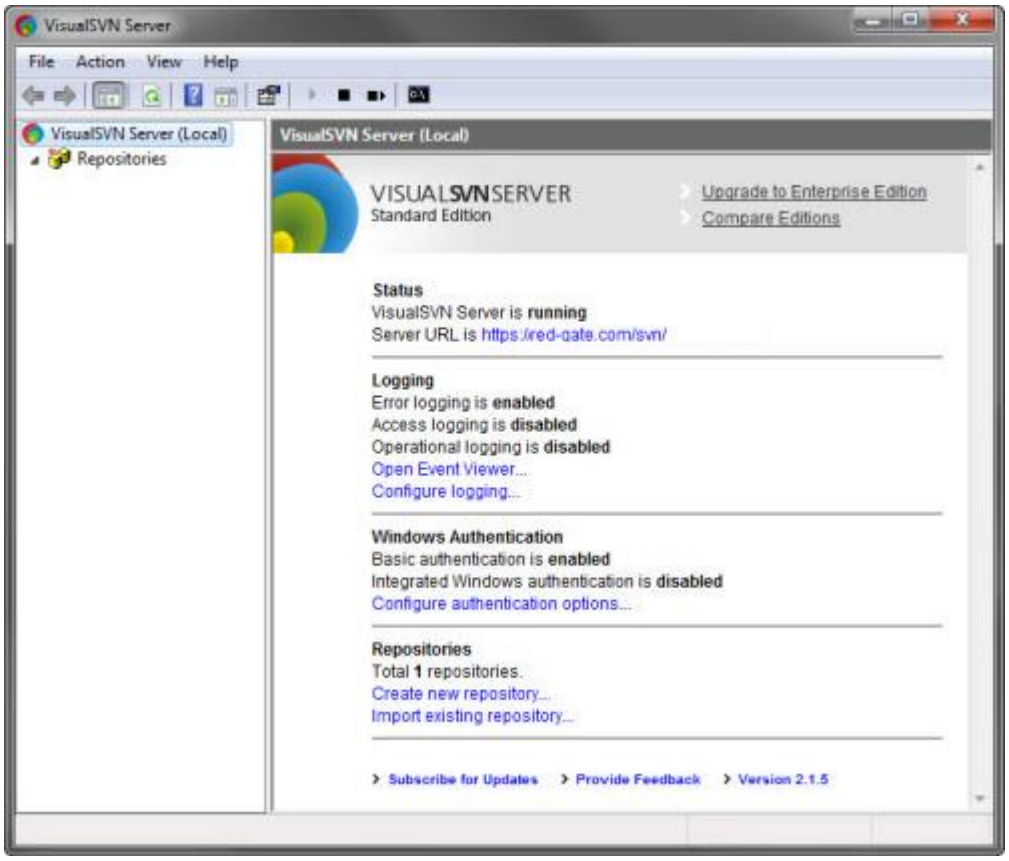
Page 4 of the installation wizard allows you to specify the location where the Subversion repositories are created, and the type of authentication:

Subversion authentication requires you to set up users and credentials on the Subversion server.

Windows authentication allows you to use your existing Windows user accounts.

Note that if you are using Windows authentication in VisualSVN Server Standard Edition (the free version), or Subversion authentication in either edition SQL Source Control may prompt you to enter your user name and password when linking a database source control.

At the end of the installation, run the VisualSVN Server Manager:



The Server Manager allows you to set up repositories and configure security.

To set up a repository to use with SQL Source Control:

1. In the Console Tree pane, to the left, right-click **Repositories**, and click **Create New Repository**

   The Create New Repository dialog box is displayed.

2. In Repository Name, type a name for the repository.

   Optionally, to create the recommended VisualSVN Server directory (http://www.visualsvn.com/support/topic/00017/) structure in your repository, select the Create default structure check box.

3. Click **OK**

   The repository is created.

Using the repository with SQL Source Control

To use the repository with SQL Source Control you will need to create a folder for your database.

To create a folder in the repository:

1. Right click the repository, select **New**, and click **Folder**

The Create Folder dialog box is displayed.

2. Specify a name for the folder, and click **OK**

The folder is created.

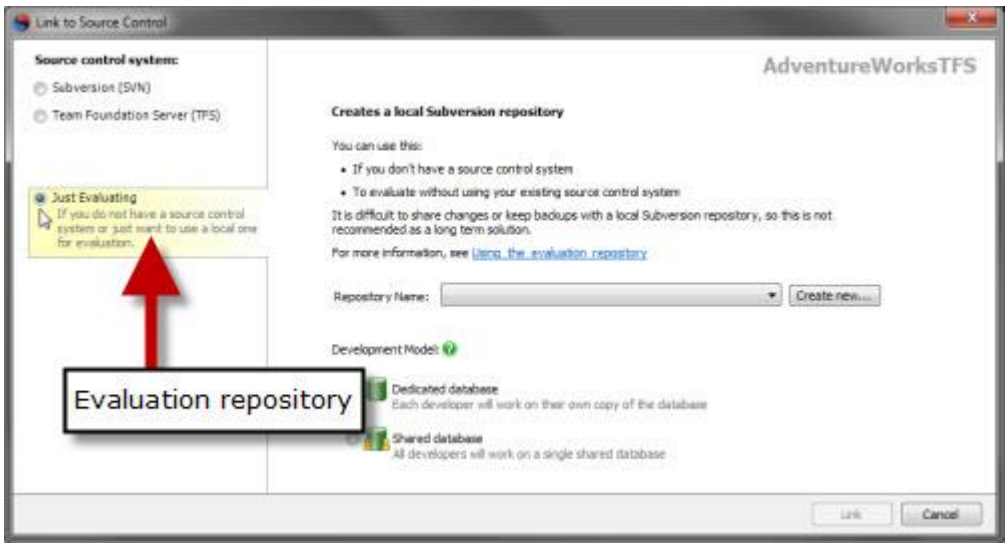
To link a database to source control (page 31), you need the URL for the repository.

To find the URL of a repository in VisualSVN Server Manager, right-click the repository, and click **Copy URL to Clipboard**

Using the evaluation repository

If you do not have a source control system set up, or do not want to use your existing system to evaluate SQL Source Control, you can use the evaluation repository.

To do this, select the *Just Evaluating* option when you link a database to source control (page 31):



For an overview of getting set up quickly, see: SQL Source Control in 5 minutes (http://www.red-gate.com/products/sql-development/sql-source-control/entrypage/5-minutes)

Creating an evaluation repository creates a local Subversion repository. You can use this to source control a database, and get set up in just a few minutes. However, local repositories have limitations.

Generally, you are recommended to set up a Subversion server (page 63) rather than using local repositories.

If your are using an evaluation repository, you can migrate it to a Subversion server, retaining the history. For instruction on doing this, see Moving the evaluation repository to an SVN server (page 69)

Local repositories are designed to be used by a single user on a single computer. It is possible to share them by giving multiple users read and write access to the repository directory. However, this is not recommended.

If you want to use the evaluation repository under a shared development model, you need to give each user permissions to access the repository directory.

A Subversion server supports the protocols *http://*, *https://*, *svn://*, and *svn+ssh://* so it is easier to set up access for multiple users, security can be stronger, connections may be faster, and you have more fine-grained control of user permissions.

For more information, see Accessing a Repository on a Network Share (http://tortoisesvn.net/docs/release/TortoiseSVN_en/tsvn-repository.html#tsvn-repository-local-share) in the Tortoise SVN documentation.

Moving the evaluation repository to an SVN server

If you have been using an evaluation repository, or any local SVN repository, you can migrate it to a Subversion server, and retain all the log/history information.

You can migrate the repository by using VisualSVN Server (http://www.visualsvn.com/server/), an installation and administration application for Subversion on Microsoft Windows servers.

VisualSVN Server is available both as a free tool (the Standard Edition), and as the paid Enterprise Edition. The Enterprise Edition includes Integrated Windows Authentication, as well as richer logging and administration tools.

This example uses the free version of VisualSVN Server.

The example has three steps:

1. Download and install VisualSVN Server (page 69)
2. Import your evaluation repository (page 69)
3. Unlink and relink the repository in SQL Source Control (page 71)

1. Download and install VisualSVN Server

To set up your SVN server, download and run the VisualSVN Server installer on your server, then follow the wizard to complete the installation.

For information on installing VisualSVN Server, see Setting up a Subversion server (page 63)

You can download the VisualSVN Server installer here (http://www.visualsvn.com/server/download/)

2. Import your evaluation repository

To import the contents of an evaluation repository to your new SVN server, you must find the evaluation repository, copy the files to your server, and then import them using VisualSVN.

1. First, locate the evaluation repository on your computer.

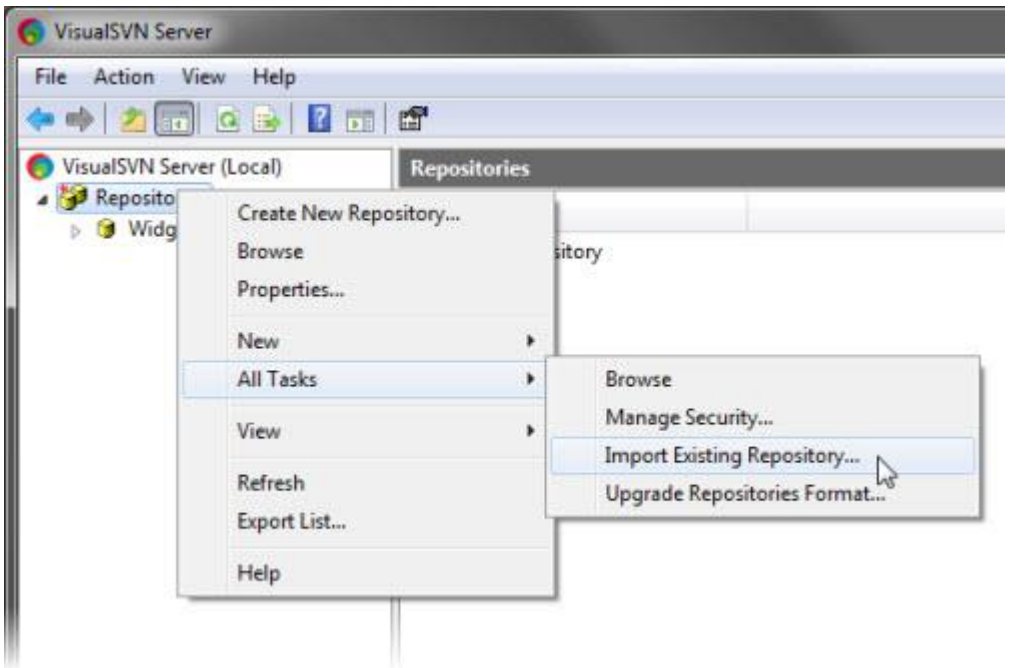
   The evaluation repository location is typically: *C:\Users\<user name>\AppData\Local\Red Gate\SQL Source Control 2\EvaluationRepositories*

   Alternatively, to find your repository location:

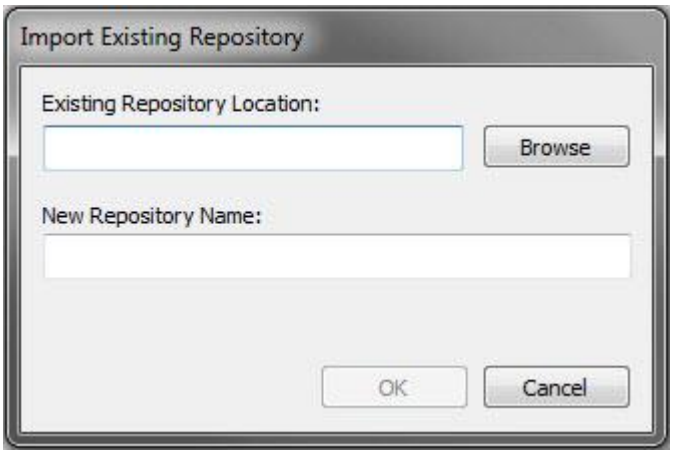
Open SQL Server Management Studio if it is not already open. In the Object Explorer, select the database you want to import, and on the Setup tab of SQL Source Control, where it says *Linked to*, right-click the red text and click **Copy**:

The repository location is copied to the clipboard.

2. Browse to the repository location, and copy the repository you want to move to a location on your server.
3. On the server, open the VisualSVN Server Manager if it is not already open, and in the Console Tree pane, to the left, right-click **Repositories**, select **All Tasks**, and click **Import Existing Repository**:



The Import Existing Repository dialog box is displayed:



4. In Existing Repository Location, paste the location of the repository copy.

   A new repository name is supplied automatically. If you do not want to use the default name, type a new one.

5. Click OK.

   The repository is created. The new repository contains any history information from the evaluation repository.

3. Unlink and relink the repository in SQL Source Control

To start using the new repository with SQL Source Control, unlink the evaluation repository and link the new one:

1. In the VisualSVN Server Manager, right-click the new repository, and click **Copy URL to Clipboard**
2. In SQL Server Management Studio, ensure the database is selected in the Object Explorer, and on the Setup tab of SQL Source Control, click **Unlink database**

   A dialog box is displayed asking you to confirm the unlink.

3. Click **Yes**

   The database is unlinked.

4. On the Setup tab, click **Link database to source control**

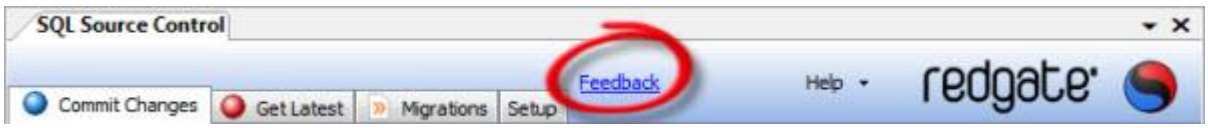
   The Link to Source Control dialog box is displayed.

5. In Repository URL, paste the URL of your new repository.
6. Under Development Model, select your development model: dedicated, or shared.
7. Click **Link**

   The database is now linked to the new repository on your Subversion Server.

Bug reports and feedback

To give feedback or make feature requests for SQL Source Control, click the *Feedback* link at the top of the SQL Source Control tab:



We are continuing to develop SQL Source Control; your feedback and error reports help us decide which features and issues to prioritize for the next release.

So please do let us know about any issues you may encounter.

If SQL Source Control encounters a serious issue, you may see an error report dialog box:

Please click **Send Error Report**, and - if possible - provide detailed feedback, as these reports are extremely valuable to us.

These error reports may include:

- a stack trace
- your SQL Server and operating system version
- the Windows user you are logged in as, and your domain
  This is so we can identify recurrent bugs.
- information from the SQL Source Control logs
  In some conditions, this may include information about your database schema.