

Revision	Description
4/4/2010	Original

## SQL-Hero SQL Source Control

### Introduction

The idea of SQL source control probably means different things to different people. If you use Microsoft's GDR for VSTS Database Edition, it could mean managing your source-of-truth in an external repository, with changes reconciled to and from it. For others, you might use scripts for all objects, maintained in an existing source control system such as VSS or SVN. Both of these approaches assume you will use a *process* that involves steps of reconciliation: your source-of-truth isn't really the database itself, so anything you do there is really just "preliminary" until it makes it into source control.

SQL-Hero's version of source control is actually a bit simpler, by design. To achieve this, it integrates directly with the SQL-Hero editor tool to provide an experience that resembles the kind of source control you might expect within Visual Studio when editing source code files. In its basest form, it makes the contents of the database the source of truth. There are a number of advantages, including:

- Other developers can see what you're actively working on and as such, prevent "collisions" (i.e. lost updates)
- It's not possible to make changes that are "forgotten" from the source-of-truth.
- There are fewer steps involved in managing what's being worked on – however, source control on its own does not manage builds per se. Other build tools within SQL-Hero can assist in that regard - or one can continue to use other tools such as GDR to deal with builds.

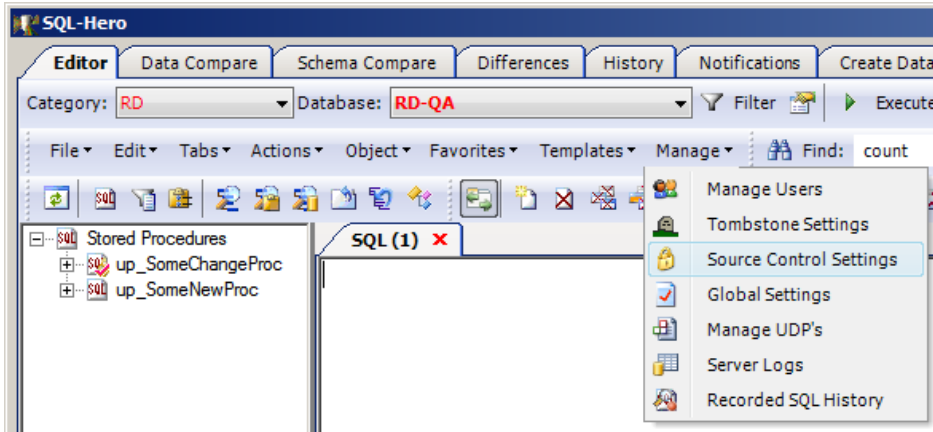
The current version of SQL-Hero supports an exclusive check-out model. This will likely change in future versions where we expect to support a merge model, plus support direct integration with TFS.

It should be noted that you do not have to use SQL-Hero source control – whether you use the editor tool, or not. You can stick with whatever process you prefer to manage change in your databases and still take advantage of other product features. For example, as noted in the whitepaper on change history tracking, it's still valid to keep an audit history of the physical database, no matter your views on source control.

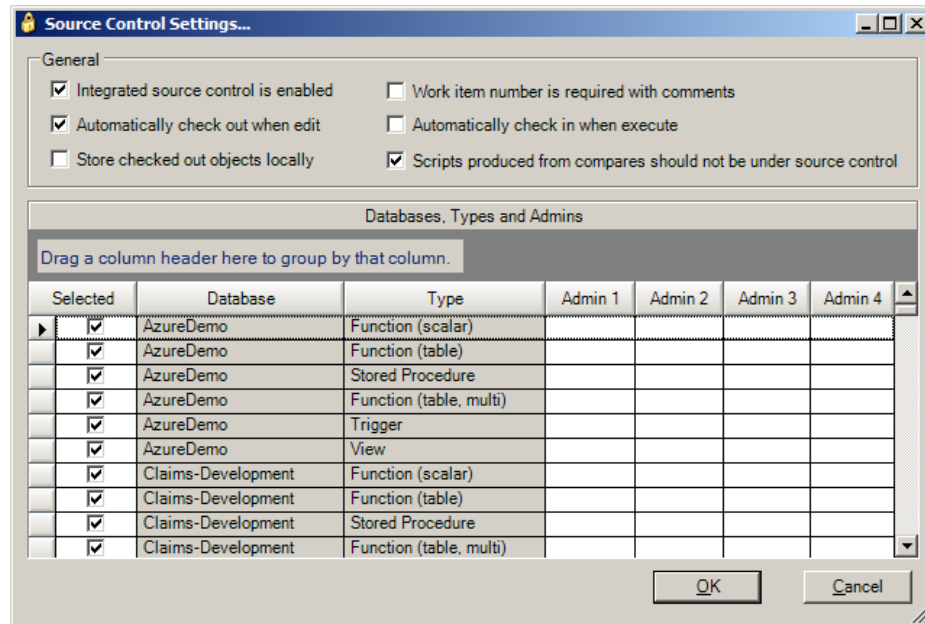
### Working with Source Control

To enable SQL-Hero integrated source control, you need to have access to SQL-Hero server components - source control meta-data is stored in the central repository. Details on how to install server components can be found in the whitepaper "Installing SQL-Hero."

On the Editor toolbar menu, the Manage -> Source Control Settings command invokes some global source control options:



Checking “Integrated source control is enabled” is the first step:



Other options include:

Automatically check out when edit	When unchecked, you will need to explicitly check out objects before they can be edited.
Work item number is required with comments	When checked, with comment prompting enabled (see below), the work item number becomes a required field.

Automatically check in when execute	Checking this means that when you commit an object by executing it, it will no longer show as being checked out to you. The time an object is checked out, therefore, is strictly the time between when you explicitly check it out (or start editing it) and when you commit its change.
Scripts produced from compares should not be under source control	The implication of not checking this is that if automatic check-outs are enabled, you can end up checking out all the objects referenced by a change script.

This screen also lets you provide the user name of optional source control administrators, by database and object type. Note that a SQL-Hero administrator is also automatically a source control administrator for *all* databases.

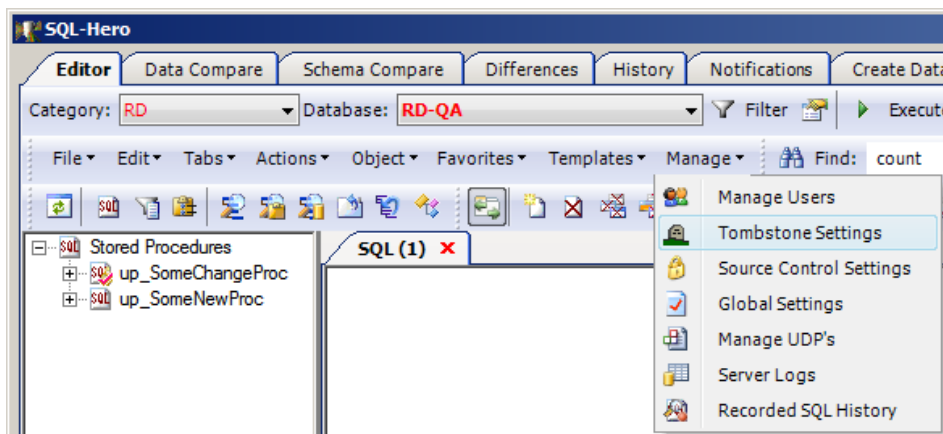
A source control administrator is allowed special privileges, including:

- Is able to undo the checkouts of others
- Is able to execute object changes despite the object being checked out to someone else

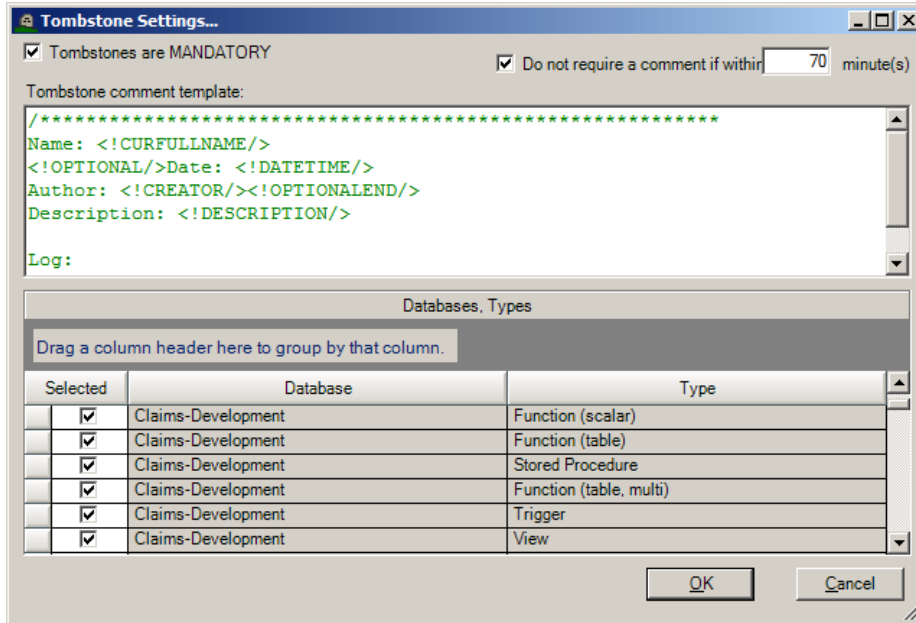
Although not strictly associated with source control, you can also force developers to enter comments when they commit object changes (and potentially check them in, if “automatically check in when execute” is checked). This compares to some source control systems which require you to provide comments on check-in. Note that SQL-Hero uses the captured comment in a two specific ways:

- Optionally places the comment inside the object itself, in the object tombstone – ensuring visibility to the comment to *all* developers, regardless of the tools they use
- Places the comment in the repository, making it available for release notes, details in notifications, etc.

This is controlled using the Editor’s toolbar menu option Manage -> Tombstone Settings:



If tombstones are identified as being mandatory, a tombstone will be added to any object that is missing one that matches the template shown on the Tombstone Settings screen. One can also force a comment, but only if one has not already been provided within a certain timeframe. This works on the premise that you may need to change an object a number of times before you're "happy" with the final product and all the changes to get to that point are logically related.



The full default tombstone template is:

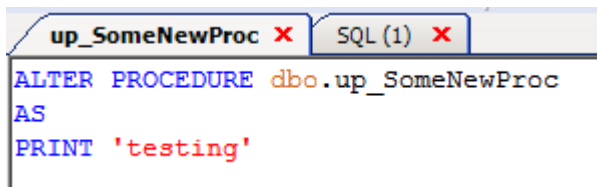
```

/*****
Name: <!CURFULLNAME/>
<!OPTIONAL/>Date: <!DATETIME/>
Author: <!CREATOR/><!OPTIONALEND/>
Description: <!DESCRIPTION/>

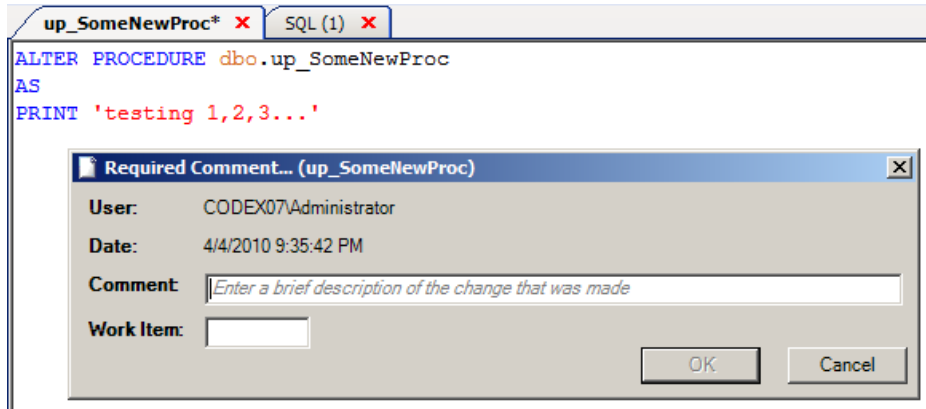
Log:
<!LOGLINE/><!USERNAME (24) /><!DATETIME (24) /><!COMMENT/><!LOGLINEEND/>
*****/
    
```

<!OPTIONAL/> and <!OPTIONALEND/> delimit regions of text that do not need to match in the existing object's text. <!LOGLINE/> and <!LOGLINEEND/> delimit the section that will accumulate comments that are added by comment prompting.

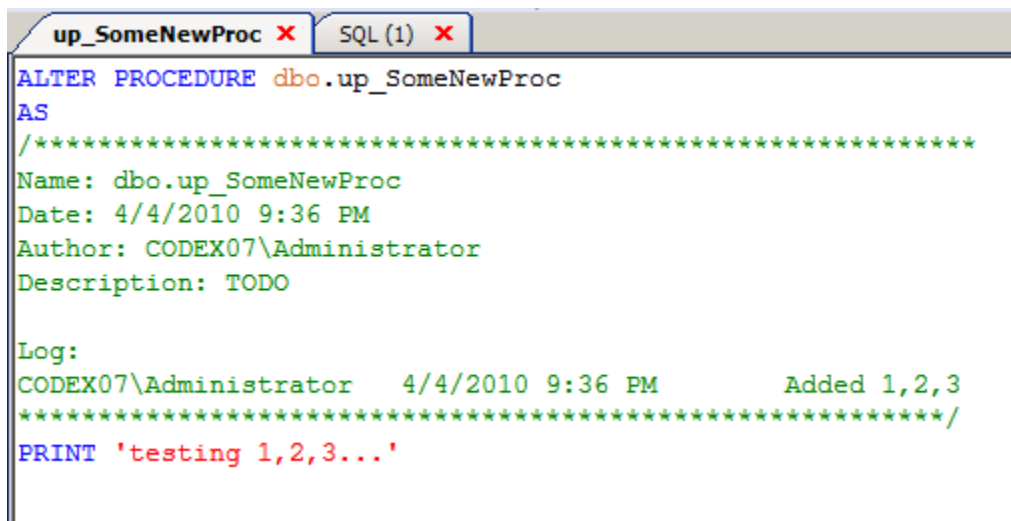
Putting it all together, assume one has the following procedure:



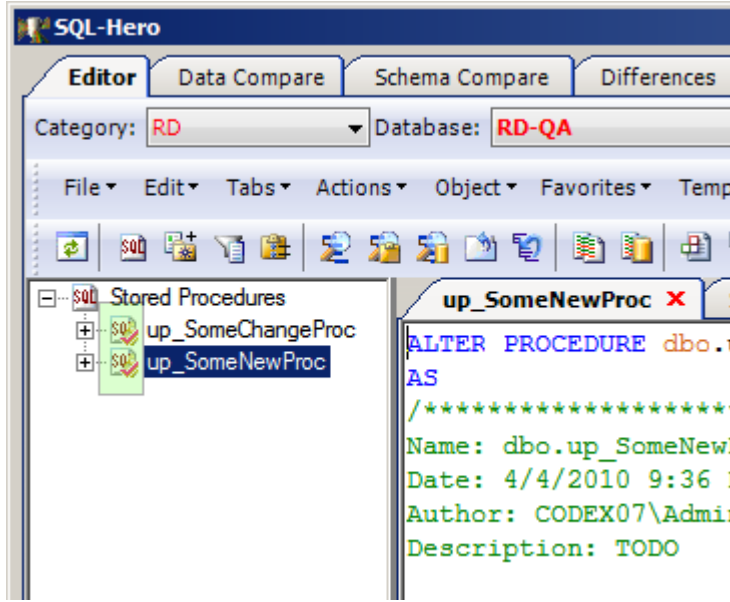
If I make a change and hit F5 to commit the change by executing the ALTER, the following appears:



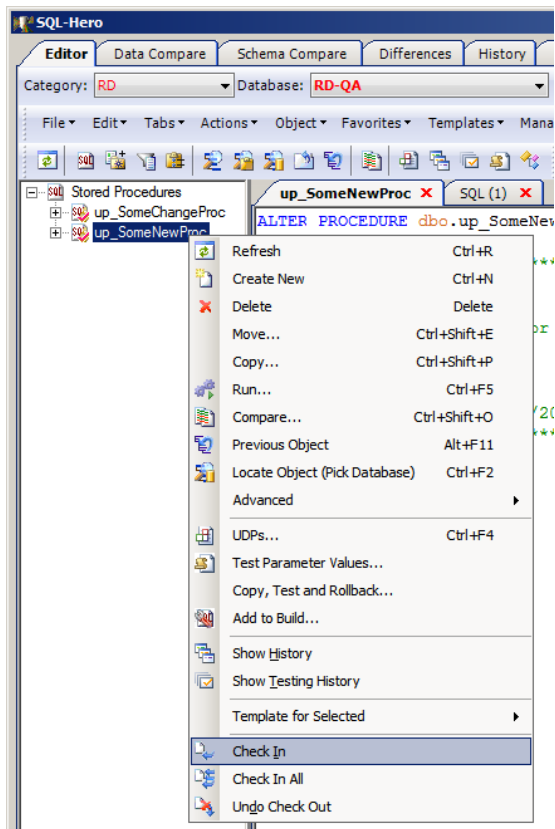
If I type in a comment of “Added 1,2,3” and click OK, the procedure is updated as follows:



Note that we have two objects showing as being checked out to me. (We could have automatic checkouts enabled, in which case up\_SomeNewProc may have started as not being checked out, but the change just made caused it to be checked out.)



Automatic check in is not enabled in this example, otherwise executing the change would have checked it in. As such, we can explicitly check it in using either the tree's context menu (right-click):



Or by using the editor's source control toolbar:



Yet another way in which you can interact with source control is using the “Pending Check-Ins” panel, as invoked from the source control toolbar:



This brings up a new editor panel that summarizes current check-out status for all objects:

A screenshot of a software window titled 'Pending Checkins'. It contains a table with columns for 'Selected', 'Database', 'Checked To', 'Object', and 'Checked On'. There are two rows of data. The first row shows 'RD-QA', 'CODEX07\Administrator', 'up\_SomeChangeProc', and '4/3/2010 07:34:03.217'. The second row shows 'RD-QA', 'CODEX07\Administrator', 'up\_SomeNewProc', and '4/4/2010 14:35:38.973'. Above the table is a toolbar with buttons for 'Refresh', 'Select', 'Toggle Selections', 'Check In Selected', 'Undo Selected', 'Show All Users', and 'Show All Databases'. A text box above the table says 'Drag a column header here to group by that column.'

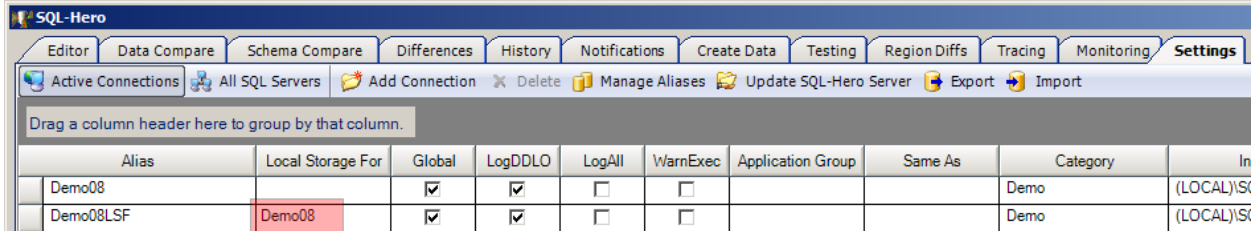
You have the option here to show checkouts by other users, and for other databases, too. You can check-in or undo checkouts on multiple objects at once from here, too.

### ***Sandbox Databases***

So far we’ve covered behavior of source control when dealing with a single database. SQL-Hero supports a variation in source control whereby two databases can be used. Here the first database serves as the source-of-truth, and the second database serves as a local “working copy.” In this case a checkout in the working copy also checks the object out in the source-of-truth. A check-in causes the working copy’s object to be applied to the source-of-truth database. This approach is well suited to when you need to work on developing an application when off-line at times: you’re able to check out objects even when you’re disconnected from the SQL-Hero application server. These are considered “provisional checkouts” and show in the object tree as being underlined. When you connect to the application server later, they become “real” checkouts. Undoing a checkout for an object in the two database model has the added effect of pulling the current version of the object from the source-of-truth database.

To set up the two database model, one needs to do two things:

- Ensure that both databases are global connections
- Name the central source-of-truth database name in the “Local Storage For” column of the working copy database’s connection, as illustrated here:



Alias	Local Storage For	Global	LogDDLO	LogAll	WarnExec	Application Group	Same As	Category	In
Demo08		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>			Demo	(LOCAL)\S
Demo08LSF	Demo08	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>			Demo	(LOCAL)\S

Here when we work with Demo08LSF, it’s the working copy. Demo08 is the central database.