Improved Database Development using SQL Compare

By David Atkinson and Brian Harris, Red Gate Software. October 2007

Introduction

This white paper surveys several different methodologies of database development, examines their strengths and weaknesses, and illustrates how Red Gate's comparison tools can be incorporated into each model. The intention of the paper is not to recommend one approach rather than another, but to describe how Red Gate software fits in to the model you choose to adopt, and improves it.

Development models necessarily take into account a number of factors that vary from organization to organization, including resources, timescales, and budget. One theme, however, is emerging with increasing prominence: there is growing recognition of the value of applying sound change management principles to the database development process. The challenge for many businesses is how to respond to this, without substantial investment or costly upheaval to existing workflows.

What do we mean by change management?

The purpose of change management is to protect the integrity of the database design throughout the development process. On a practical level, this means the ability to identify what changes have been made, when, and by whom; and a means to scrutinize and – where necessary – undo individual modifications. Another common requirement is the facility to access and distribute the most recent iteration of a database schema under construction at any point during the development cycle. In an environment where a number of developers work in parallel to edit a database design, integrating their activities can prove difficult without a clear set of agreed processes.

A source control system is typically the critical application around which a change management regime is structured. The source control application enforces good practice by providing a central repository for all work on the database to be stored in on a regular basis, and offers features such as security, file histories, merging, branching, labeling, auditing and reporting, that can be utilized within a project to offer a greater degree of control, and to minimize risk.

This still poses the question of how to actually edit and save the database design. Making changes to a database does not automatically leave any permanent record of what was altered, and how. A tool such as SQL Compare® allows different versions of the database schema to be synchronized and, in this approach, while the schema itself may only ever exist as a live database, the means to upgrade it to a modified version is saved permanently as a set of SQL commands in a script file. To save the schema itself, snapshots may be taken at agreed milestones; these can be compared with the current database to identify new or edited objects, or even synchronized against an empty database to generate individual object script files that will rebuild the saved schema.

Where does Red Gate SQL Compare come in?

Red Gate's schema and data comparison tools unquestionably save the database developer time and effort. They offer solutions to some traditionally problematic issues, particularly, how to migrate database changes and synchronize development activities within a potentially rapid and dynamic development cycle.

However, the intention of this paper is to demonstrate that these tools can also be used as part of a more rigorous, controlled development process, built around change management principles and the discipline inherent in using a source control system.

SQL Compare 6 Professional is a new release from Red Gate that gives the developer the option to work on a database schema at the level of individual database objects. A database schema can be saved as a set of SQL object creation scripts, which can be synchronized with a live database; this provides a more granular option for storage of the database design, and offers a very economic and simple solution to the issue of source control. This paper illustrates how to incorporate the new features of SQL Compare 6 Professional into a workflow that maximizes the benefits of working with files in the context of a source control solution.

It is hoped that what emerges is the flexibility of Red Gate software. SQL Compare 6 allows you to work in whichever way is considered most appropriate within your business and does not enforce any one particular workflow template.

In this paper, three common development models are discussed.

MODEL 1 – The ad hoc merging model

The ad hoc model is so called because modifications to a database design are coordinated at specific intervals, rather than as part of a more methodical system of recording each change to every database object. Periodic merges of modified developer database instances are carried out to synchronize each developer's schema with all the others so that, at milestones in the development cycle, all the database instances are the same. At this point, a snapshot may be taken for version control.

Process

1. Before development begins, restore the live production database to one or more SQL Server developer instances, so that required modifications can be carried out and validated for each developer against a real, isolated or "sandbox" database.

2. Develop the database on each local machine. Use a tool such as Query Analyzer or Management Studio to make changes to the database.

3. When a new build is required (e.g. for a Test or UAT environment) and if more than one developer has been involved with the changes, use SQL Compare to synchronize the development databases. A nominated developer will need to coordinate this process; each modified instance of the database should be compared in turn against a copy of the last production version of the schema, and the changes assessed and incorporated incrementally. In SQL Compare, each object that differs between two development databases can be marked as a change (to keep) or a new or modified object (to inherit from another developer) so, in this way, synchronization is possible between several concurrent developing versions of a database. This produces the new database version.

4. Produce a snapshot from the new database version, and save it into a source control system, if required. This allows a database to be synchronized to a specific, previously-saved build.

5. Use SQL Compare to compare this new version of the database with a copy of the production database (it is recommended that you use a staging database at this point, rather than the actual production database), and generate a synchronization script.

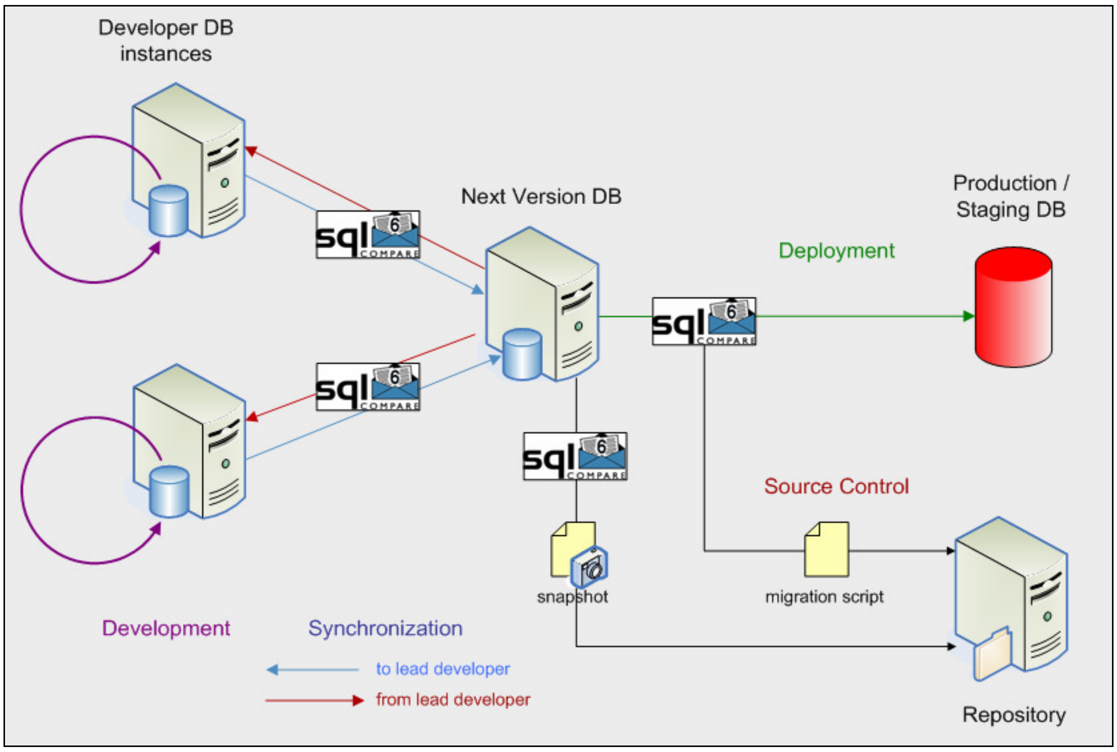
6. Validate the synchronization script by applying it to the staging database. After testing it and making any appropriate modifications, save it back into source control. Label the source if appropriate.

7. The developers continue development until the next build is required, and the process is repeated.

8. Take a backup of the production database before running the synchronization script.

9. Run the synchronization script on the production database.

Process Diagram



Advantages

This development model allows a small team of developers to work collaboratively on a single database project, with each developer having their own isolated instance of the database.

As the developers are working on a real database, they can validate their modifications as they develop.

No manual editing of scripts is required. Upgrade scripts and database creation scripts do not have to be maintained manually.

Disadvantages

There is always the potential for merge conflicts when developers are working on instances of the same database. If two developers update the same object, then there is the possibility that modifications may be inadvertently lost during synchronization. Careful use of SQL Compare can reduce this kind of error to a minimum but, ultimately, developer coordination and scrutiny are required.

Concurrent development with ad hoc merging requires a degree of coordination to ensure that development database instances are periodically synchronized, and also to version the schema when required.

There is no source control at the object level. If bugs have been introduced, it may be difficult to identify exactly when the problem first appeared, and rolling back to a previous snapshot without manual intervention will undo other modifications.

Conclusions

The ad hoc merging method is good for small to moderately-sized development projects, where source control may be less of a consideration, due either to budgetary constraints or to the database simply not being very complex. SQL Compare makes this development approach quicker and simpler, as it allows developers to merge their work without the need for painful scrutiny of hundreds or thousands of lines of SQL. It does allow for a basic level of schema versioning, providing schema rollback ability at intervals, but not at the level of individual objects.

MODEL 2 – Object level source control

Object level version history and labeling is considered by some to be the basis of database change management best practice. It allows you to place your source control system at the heart of the development process, and provides a means to control and track changes to your database design in a detailed and meticulous manner. The new object creation scripts functionality in SQL Compare Professional 6 means that you can now work at the level of object scripts, and still use SQL Compare to compare and synchronize your database modifications.

Process

1. Use SQL Compare 6 to export the production database schema to a set of SQL object creation scripts. You can specify the folder structure to use, so that each object type is contained within a different folder. Organizing your objects into separate folders will later help to locate objects that have been modified.

2. Add the object scripts to a source control system. This will ensure that you can use the features of the source control system to manage the files, for example, to check them out and in when changes are made, and to view the history of an individual object.

3. Before any modifications are made to the database schema, label this set of object files as the baseline or last milestone version.

4. Each developer obtains the latest version of the object files from the source control system, and uses SQL Compare to ensure that the schema on their local instance is synchronized with the version under source control.

5. Each developer updates the database on their local machine (isolated SQL server instance) using Query Analyzer or Management Studio. When a change is made which is tested (and approved if appropriate), the developer synchronizes to the scripts folder. Finally the developer checks in any new and updated script files. Some object scripts will now be identified in source control as being a later revision.

6. At frequent intervals, each developer ensures that they have the latest set of files from the source control system, by using "get latest version" (if using snapshot-based source control) to obtain object files, and using SQL Compare to synchronize to their own database instance. All checked-in objects, irrespective of which individual made the change, will therefore be propagated to all development machines.

7. When required, the set of object files in the source control system is labeled as the next version, and synchronized against the staging database to generate an upgrade script using SQL Compare. Make any necessary modifications or fixes to the upgrade script, then use it to update the staging database.

8. The developers continue development until the next build is required, and the process is repeated.

9. Take a backup of the production database before running the synchronization script.

10. Run the synchronization script on the production database.

Setting up your source control application

Using SQL Compare 6 Professional to synchronize a database schema saved as a set of object creation scripts means that files corresponding to objects that have been updated will be modified. A list of these files is displayed before the synchronization is confirmed.

How you edit files that are stored in a source control application depends upon the methods your particular tool supports.

Edit-Merge-Commit method

Using the "edit-merge-commit" method of source control (also sometimes referred to as CVS style) allows files to be modified on your local machine without having to specifically check files out of a repository or make them writeable prior to the edit.

1. Run the synchronization, with the scripts as the target schema, using SQL Compare 6 Professional. There is no need to review the files that will be edited or created at this stage, as these can be identified by your source control client.

2. Files corresponding to deleted objects are not deleted during synchronization. These files must be identified individually, and removed from source control. Typically they will be empty, and therefore around 0 KB in size.

3. Using your source control application, identify any modified files in the designated folder and commit them to the repository. If there are files to be deleted, remove these first. If other developers have been editing the same files, a merge will be required before the files can be saved back to source control. Additionally, search for any new files (representing objects that did not exist in the target schema prior to synchronization) and add these files to source control.

Checkout-Edit-Checkin method

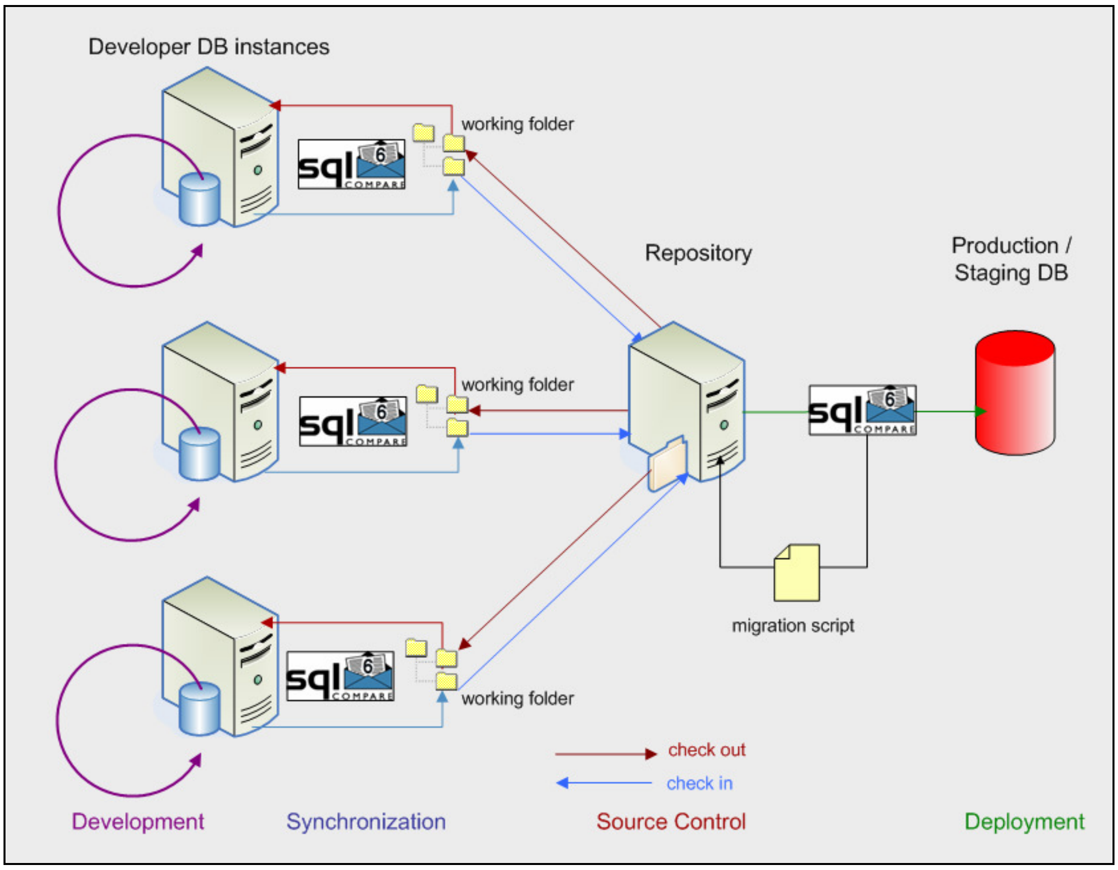
Using the "checkout-edit-checkin" method requires that files to be modified must be made writable before you run the synchronization. You can either check out just the files that you know will be updated (listed as part of the synchronization process) or, if you have SQL Compare Professional 6.2 or later, you can use SQL Changeset to automate file checkout on synchronization.

1. Before confirming the synchronization, review the list of object script files that will be modified.

2. SQL Compare Professional will check out the relevant script files. At this point, you may want to ensure that the files to be modified are up to date in your local folder, that is, they match the version stored in the main repository.

3. Following the synchronization, you can use SQL Changeset to commit the script files back to source control, and add any new files.

Process Diagram



Advantages

Each individual schema object script exists as a separate file under source control, allowing the history of any object to be tracked as the schema evolves in the project. In the situation where unwanted changes may have been included in a version of the schema, the affected objects can be reverted to their last correct version, leaving the rest of the schema intact.

Developers still develop on a real database, allowing them to validate changes and ensure they are proving the integrity of the database schema as they proceed with development.

Source control features can be utilized to aid project management; for example, the use of comments when checking in each script, or to identify which individual has altered a particular object.

When you are working as a team on simultaneous development of the same schema, the source control application can be set to enforce exclusive checkouts, reducing the risk of changes being overwritten or lost.

Merge situations (parallel development on the same objects) can be more closely controlled, by dictating that the two conflicting versions of the script file must be merged before being checked in.

Disadvantages

Use of source control requires a more disciplined and procedural approach, which may be unfamiliar to some developers who prefer to work in a more dynamic regime.

Source control applications require administration, management, may take time to set up, and can slow down projects when there are problems with the host server. Purchasing and integrating new software may be expensive.

Synchronizing to scripts using SQL Compare 6 Professional may require some manual interaction with the source control application, in order to obtain the latest file versions, and to commit modified scripts. This however will be resolved in an early point release of SQL Compare 6 (see Conclusions, below).

For smaller projects, where there may be only a single developer, source control may be unnecessary and overly time-consuming. Individuals on the project have to be trained to use a particular system in operation, and to follow the correct processes.

Conclusions

The object level source control method offers an excellent solution for database development that adheres to the principles of trackable change management, while still allowing developers to apply changes directly to a real database. Using SQL Compare 6 Professional allows a database schema to be stored within the security of a source control application as a set of SQL creation scripts, and for these scripts to be compared with a real, modified database and updated with changes. In this way, changes to the schema can be controlled and audited in a detailed and comprehensive manner, at the level of component objects, without any requirement for scripts to be edited manually off line. Future versions of SQL Compare 6 Professional will offer direct integration with common source control applications, making this model even more efficient.

MODEL 3 – The off-line development model

The off-line development model is characterized by how the schema is edited. The previous two models describe how each developer works on a real, isolated instance of a database that was, at some point, a duplicate of the production or staging database. This allows for instant feedback as to whether each change they are making to the database structure is valid. Only when the changes are validated are they synchronized to a coordinating developer for periodic merging (Model 1) or saved back to a scripts folder for placing in a repository (Model 2).

In this third model, the schema is developed by directly editing the SQL within the relevant creation scripts. These edits can be made in any text application; no real database exists until the scripts are executed to create the database schema. This offers a degree of freedom in allowing development to take place off line, with no requirement for access to a database. Typically, however, validation against a real database is still required periodically, to ensure the changes are accurate, before the new schema is finally deployed.

The ability, in SQL Compare 6 Professional, to compare a set of SQL scripts with a real database schema, provides a means to validate the schema against a test database, and to deploy the changes subsequently to the production/staging database.

Process

1. Before development begins, obtain the latest version of the schema object scripts from source control, and check them out to the local working folder.

2. Edit the files in the working folder directly, using your preferred text editing tool.

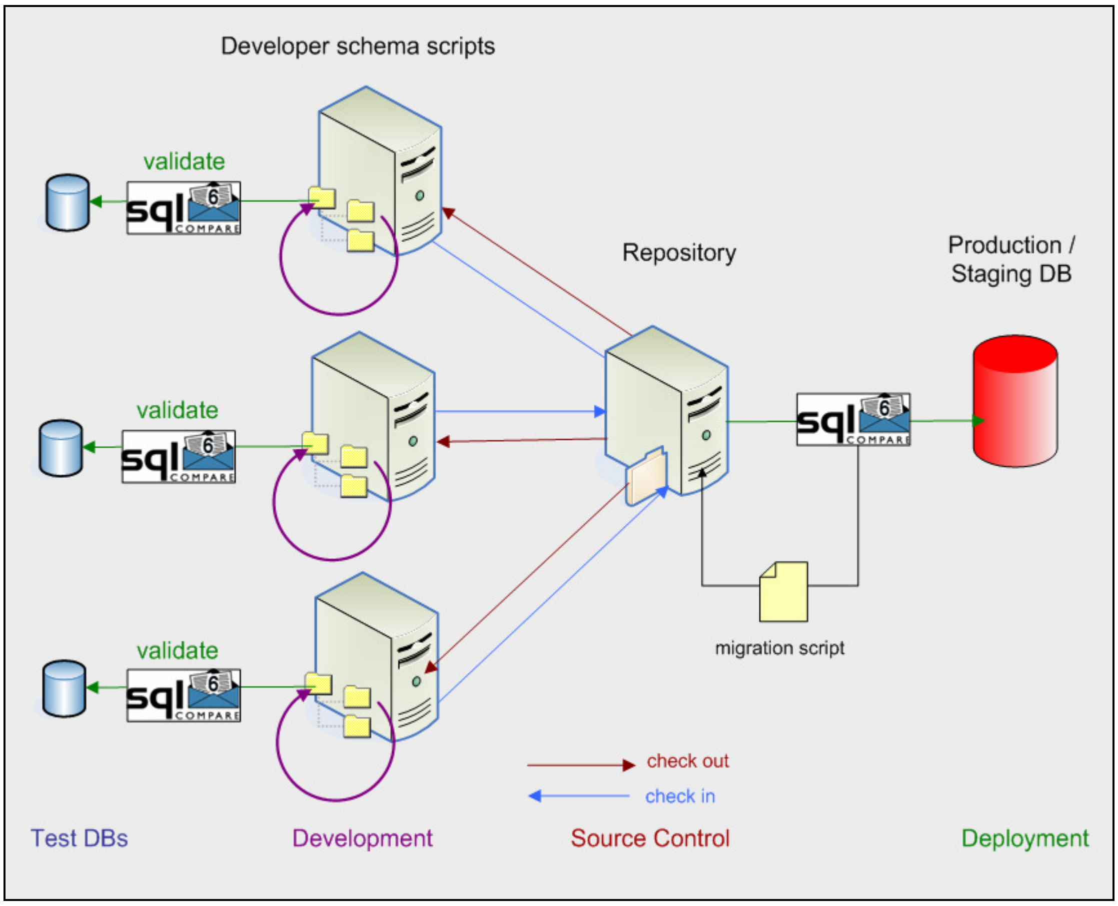
3. Deploy the changes to a developer instance to validate the modifications. Use SQL Compare to view changes between the edited scripts and the test database, and synchronize the test database if required.

4. Check the modified files back in to source control.

5. When the new version of the schema is complete, use SQL Compare to compare it with a copy of the production database (it is recommended that you use a staging database at this point, rather than the actual production database), and generate a synchronization script. This script can be checked and validated, and then stored in source control, before being later run on the production database itself.

6. Validate the synchronization script by applying it to the staging database. After testing it and making any appropriate modifications, save it back into source control, take a backup of the production database, and finally run the script against the production database.

Process Diagram



Advantages

Editing a set of script files allows more direct interaction with source control, especially when using an editor with integrated source control functionality.

This process reduces the risk of objects being modified by two developers at once.

SQL Compare 6 Professional ensures that the object scripts are executed in the correct order, taking into account dependencies between them. This negates the need to manually maintain a file that lists the order in which scripts should be run.

Disadvantages

There is no immediate validation of the schema integrity, as changes are being made on script files rather than a real database.

Working off line means forgoing the productivity gains of tools such as SQL Prompt™ and SQL Refactor™ which are specifically designed to work with real instances of databases, and thereby ensure the validity of the database, as it were, in real-time, at the point of change.

To test changes to the database, the script files must be deployed to a real database, which adds an additional step.

Conclusions

Off-line development is considered by some to be the purest form of database development, as changes to the schema are always made by directly editing code. Change management processes can be easily applied to this model, as the schema exists primarily, if not always exclusively, as a set of files that are saved within a source control application. Editing the script files is synonymous with changing the schema, so it is usually straightforward to determine the status of the project at any given time.

The more problematic aspect of this model is how to test the integrity of the schema and apply updated objects to the production database. A set of scripts may be syntactically correct, but tracking the dependencies between them is required, to ensure that the schema is built in the correct order. SQL Compare 6 Professional can read from, and synchronize to, a set of script files, taking all dependencies into account. This minimizes some of the potential difficulties with this model – schemas can be compared between real databases and stored script files. The lack of instant validation to schema changes places more responsibility on the developer; some process is usually still required in which the schema is built and tested regularly.

Summary

This paper reviewed three different processes for developing and maintaining database schemas. Each approach has different strengths and weaknesses.

Model 1 offers potentially the most rapid development cycle, allowing developers to modify isolated database instances directly and then synchronize their modifications using SQL Compare. However, this process provides limited opportunity for granular source control.

Model 3 involves editing the schema as a collection of SQL creation scripts, which can be stored in a repository and each file managed individually to monitor object level changes. Issues concerning how to regularly build and test the database design may be more problematic in this model.

Model 2 exists somewhere between the two other methodologies: it enables direct editing and testing of live database instances, while also offering the potential for storing the schema in source control as a set of object scripts, and comparing a live database schema with saved script files using SQL Compare Professional.