Database deployments

Choosing a state-based or migrations-based approach
Executive summary

In application development, it is increasingly common for code changes to be automatically integrated and deployed using Application Lifecycle Management (ALM). This system-wide process means multiple teams can work on the same project, errors are reduced, and development is speeded up.

Similarly, to achieve reliable, repeatable database deployments as part of Database Lifecycle Management (DLM), it should be possible to reconstruct any version of a database from the scripts and objects in the Version Control System (VCS). Every database build and change, however trivial, should therefore start from version control, ideally alongside the application code.

The speed and volume of changes to modern software systems means managing database changes manually is risky and error-prone. Thankfully, there are a growing number of software tools to automate the process. At a high level, there are two methods for developing and deploying databases: the state-based approach, and the migrations-based method.

**The state-based approach** aims to enable a single step between the current database state and the desired database state.

The approach uses individual table-level scripts and, each time a table is modified, the latest script is committed to a VCS. In effect, the current state of each object in the database is versioned. Tools used to migrate a database from one version to another compare the source to the target and auto-generate a script to synchronize the two states.

**The migrations-based method** focuses on capturing lots of individual changes and delivering multiple small scripts to effect larger, iterative database migrations.

In this instance, one object change script per ALTER command is generally used to migrate a database progressively from one version to the next. The tools that migrate a database via the change scripts simply need to apply each script in the correct order.

The key point to remember is that both approaches offer benefits in their own right, and there isn’t a right or wrong method — what works best depends on the needs of your team and organization.
Contents

The state-based method 4
  The challenges 4
  The opportunities 5
    Tracking changes 5
    Making error-free comparisons 5
    Creating custom migration scripts 6
    Automating database deployments 6
    Providing a dynamic working environment 7

The migrations-based method 7
  The challenges 8
  The opportunities 8
    Working with your VCS 9
    Tracking changes 9
    Automating releases with Octopus deploy 10

Choosing the right method 10

Summary 11
The state-based method

The state-based method of database development compares the state of the target environment such as testing, QA or production with the state of the development environment. It then determines what changes need to be made to get it to match development, and produces a script to effect the changes which is committed to your version control system.

Such an approach gives developers and DBAs alike ‘at a glance’ access to the current state of every object, while making it easy to audit the change history and identify who added or removed code, and when they did it.

Those changes are also easy to understand and follow because the VCS will always hold the latest build script for each object which, when committed, will be documented and liberally-commented.

Take CREATE scripts for tables, which can be complex. There are many subordinate objects that have no existence without the table, such as constraints, indexes, triggers, and keys. A well-documented script makes it easy for developers to see the exact state of an object at any given time, and understand how changes are likely to affect other objects.

At the comparison stage, a tool will typically check the database state in every environment before applying the changes. For situations where database changes are applied in several different ways, this provides a high degree of confidence changes will be executed correctly.
The challenges

Use of a source control system has long been an established part of the development process for application code but not, sadly, for database code. Largely, this is because, unlike application code, database code is all about persistence. You can't simply recompile the database into a new form: you must take into account the data and structures that already exist.

Mechanisms for automating the process of getting a database into source control, and deploying it from that location, have often either been manual or home-grown and difficult to maintain. However, tools like Redgate’s SQL Source Control now automate this and make it very straightforward to follow a state-based approach.

Some developers worry too that state-based tools can appear opaque in their operation due to their reliance on a comparison engine. However, many users work around this by reviewing scripts prior to deployment via a tool such as SQL Compare.

Similarly, where a comparison engine can't work out the changes correctly in, for example, a table rename, users might worry about the comparison route because it is regarded – incorrectly – as an ‘all or nothing’ approach. There is an assumption that a comparison tool will drop the existing table – and its data – and create a new table instead of carrying out the actual rename operation.

The migrations feature in SQL Source Control lets users accommodate this by creating a migration script containing additional SQL to specify the intent of the change.
The opportunities

There are a number of key areas in the state-based deployment process where third party tools can help automate and ease development.

Tracking changes

Tracking changes using a VCS is standard practice in application development. Despite the advantages it can also bring to database development, many companies are wary of employing it because they think it will introduce unfamiliar processes.

Redgate’s SQL Source Control integrates directly into SQL Server Management Studio (SSMS) and links and synchronizes the database to any existing VCS like TFS, Git or Subversion. Database developers can seamlessly get and commit changes, share changes to schemas and reference data, check a history of database development, and roll back changes, all without leaving SSMS.

Teams can thus be reassured that the changes to the database are held in the same VCS repository and are in sync with changes to the application.

Making error-free comparisons

At the comparison stage, it is vital the comparison engine itself is trusted to do the job with zero errors, and can be tailored for specific use-cases.

Redgate’s SQL Compare is the industry standard and is used by 71% of Fortune 100 companies because it is easy to use and has various options for including or excluding different kinds of database objects (procedures, functions, tables, views, and so on). Alongside it, SQL Data Compare supports data comparisons, allowing reference data (such as store locations or product codes) to be updated in the target database.

The tools also have command line support, allowing them to be triggered and controlled by scripts and processes which have been finely tuned to suit the individual needs of different users.
Creating custom migration scripts

For schema changes, SQL Compare offers a simple, robust, and proven method. Like all comparison tools, however, there are some changes that a comparison engine can’t work out correctly. So, coming back to the earlier example, where a table needs to be renamed, a migration script is required to deploy the change.

In this instance, the migrations feature of SQL Source Control is invaluable. It gives users a simple, repeatable way to create custom migration scripts, define how existing data is handled between changes, commit them, and use them in database deployments.

Automating database deployments

DBAs and developers who prefer making database changes in SSMS haven’t had an obvious, easy path to get changes into version control and, from there, into their Continuous Integration (CI) and automated deployment processes.

As has been seen, SQL Source Control plugs into SSMS and links databases to existing version control systems. DLM Automation then works on top of SQL Source Control and plugs into standard CI and deployment tools. Users can automatically trigger CI processes upon check-in to SQL Source Control and set up automated deployments through, for example, Octopus Deploy.

Providing a dynamic working environment

In new or prototype projects, the state-based approach comes into its own. Where v1.0 of a database is being developed, for example, there are typically many developers and frequent conflicting changes.

Because there is nothing to preserve in Production at this stage, the state-based approach can accommodate twists and turns in the schema design simply by dropping and recreating the database whenever required.
The migrations-based method

The migrations-based method tracks each change to a database in a separate script and then executes each script, in order, on the production database, to move it iteratively from one known state to another.

In other words, when developers update their working folders from the VCS, they don’t retrieve the latest versions of each object, as with the state-based approach, but instead a series of scripts they must run, in the right order, to progress the database from one version to the next.

It thus provides the greatest level of granularity for database change management because users define change scripts upfront to have full control over how changes are deployed. Where automation is required, tools like Redgate ReadyRoll make it simple to run automated tests and deploy artifacts across environments.

For changes involving tricky data migrations, the change scripts should include extra logic that defines how existing data is handled between changes in the organization of tables. Each change script should also contain the necessary logic to ensure it is idempotent, in that it will have no additional effect if it is run more than once on the same database.

Once again, migration tools such as ReadyRoll handle this automatically, making it unnecessary for users to handle the logic themselves.
The challenges

Unlike object-level scripts, migrations-based scripts must be treated as immutable once they are first successfully used, because alterations or breaks in the script chain will render the migration unusable. Instead, extra scripts are added to perform further schema changes or data updates. This helps to ensure any database can be upgraded without needing to inspect it first, as long as the database objects have been changed only via the migration tools.

While the migrations approach means granular control can be exerted over database modifications, it works against some of the standard source control processes. If every table change is a new immutable script, there is effectively nothing to version, and the diff capabilities of the VCS become redundant.

ReadyRoll mitigates this with an Offline Schema Model that provides a read-only view of the schema. This is kept in-sync with the database whenever a new script is generated, allowing the tracking of object changes in a manner identical to the state-based approach.

Another caveat is that most migrations frameworks assume database changes will only be made through the migration tool itself, recording each migration in a metadata versioning table. Where changes are made directly in the database or via another method, migration tools can be a complication.

Tools like ReadyRoll cater for this by allowing a drift correction step to be included in the CI process that identifies if such changes have been made.
The opportunities

For small or simple databases, hand-rolled scripts work well enough and you have confidence you are in full control of the SQL scripts and therefore the database. As databases grow in complexity, however, or other colleagues or teams need to change the database, managing the deployment of a large number of scripts can be a cumbersome task. This is where there are many advantages to adopting a third party tool.

Working with your VCS

Developers typically use an Integrated Development Environment (IDE) like Visual Studio to develop application code, together with a VCS such as Team Foundation Server (TFS).

Redgate ReadyRoll is an extension for Visual Studio that allows developers to deploy database code alongside application code, sync schema and data changes using the existing VCS, and automatically create numerically ordered migration scripts. It offers a pragmatic workflow for making database changes that combines making changes directly to a database with sequential SQL scripts.

It does this by inspecting online databases and generating the sequential scripts in the background, using the SQL Compare engine to compare and deploy SQL Server database schema quickly and accurately.

The numerically-ordered SQL scripts are added to a project within Visual Studio and can be refined as needed before being checked into the same version control system used for applications. Importantly, ReadyRoll uses semantic versioning to keep those scripts organized. By placing scripts into independently enumerated folders, developers can reduce the noise in their database projects and focus only on the scripts they need.

This makes it particularly suitable for working with existing databases that may not have been created programmatically and also for working in mixed teams of DBAs and developers, where some people may make changes directly in databases and others may write SQL scripts.

The Programmable Objects feature also allows users to deploy stored procedures, functions, and views with re-usable files. Rather than having to add a new migration every time a change is needed, the Programmable Object script can be edited to re-deploy the database object.
**Tracking changes**

When running a migration, ReadyRoll checks against the metadata version table to see which script was last run, and then runs only those scripts that are numbered sequentially after that version, in the order provided.

By storing additional information in the log table such as a description of the migration and the author, the script also provides very useful traceability for database changes, which can be essential for regulated industries such as banking, finance, or healthcare.

**Automating releases with Octopus Deploy**

Once a ReadyRoll project is checked into source control, ReadyRoll can build Octopus Deploy-ready packages, allowing the database to be deployed along with applications to Dev, Test, or Production environments.
Choosing the right method

As has been outlined, there are different factors influencing how database changes can be handled. Many are technical, some rely on a team’s capabilities or preferences such as working in a particular IDE, others relate to the phase of software development.

There are advantages, for example, in having the latest state of the database schema in version control, in the form of object-level scripts. This is a natural fit for standard application version control processes, and provides an audited history of changes and fast insight into the current state of every object. This does, though, rely on the auto-generation of schema synchronization scripts to apply a set of changes to the target database.

Similarly, when data in a database needs to be updated as well as the schema, the migrations-based approach handles the task with ease. The larger the database or development team, however, the more unwieldy it becomes.

While a single, unified approach to database change is clearly desirable, it’s important to recognize both approaches may be needed for different parts of the system, or at different times in the development phase.
Summary

There is no rigid, fixed approach to database deployments. Both state-based and migrations-based approaches have advantages and disadvantages in different contexts and at different phases of software development.

It is therefore important to understand which tools and approaches to use in a particular situation in order to avoid unnecessary conflicts and confusion. Redgate offers tools for both, giving users the flexibility to choose the one that best meets their needs:

- SQL Developer Suite (containing SQL Compare and SQL Source Control) for state-based database development
- DLM Automation for state-based database deployments
- ReadyRoll for the migrations-based method

Visit www.red-gate.com for more information.