# State v Migrations based Database Development

There are two philosophically different ways of dealing with database development. These are state based database development and migrations based database development. Each of these work well and can be used for most of your database development. Each also has flaws or limitations that you may need to work around.

In the next two sections, we’ll look at how each of these two development models works, along with the advantages and disadvantages that you may face when using either of them.

Note that we will explore the issues of development related to relational databases that typically allow various types of code inside of the platform. Various other types of databases, such as document databases, graph databases, etc., are not discussed in this course.

# What is State Based Migration?

A set of database code exists in a state at any point in time, just like any application code. However, unlike application code that can be completely removed and rebuilt, parts of a database must be transitioned from one state to the next. This is true of tables, but potentially other objects in relational database systems.

As a result, we cannot just compile the new version of code in the database. Instead, we need to provide transitional code that will change the state of the code.

In a state based development system, we use some sort of comparison method to examine the current state of a database (or code in a version control system) and compare that to another version of the database (or version control), and generate the SQL code required to migrate objects from one version to the next.

The typical flow for this is to make changes in development and then perform a comparison with the version of the database in a downstream environment, such as production, and then generate a script to transform the downstream environment to look like development. In essence, we generate a script to “deploy” our code. Let’s look at an example.

Demo:

Let’s show a table, let’s make changes, let’s compare.

In the demonstration, I used the Schema Compare from Visual Studio, but there are many ways to perform this comparison. A number of third party companies, including my employer, Redgate Software, make comparison tools that allow you to compare two versions of a database and produce a script to deploy the changes. These tools make it much easier to determine what has changed and produce a script.

Advantages and Disadvantages of State

There are good things with using state. One is that developers can make changes, undo them, make new changes, and never worry about tracking those changes. When the developer feels that development is complete, they can run a comparison tool and determine what changes need to be sent to the downstream database. This simplifies database development as it occurs.

The other advantage is that any changes which are made in development and then discarded are never transferred to downstream environments. Since the deployment of changes can take time, and sometimes significant time, this speeds up the deployment of database changes. I have worked with some schema changes that take minutes or even hours to complete on large tables. Avoiding making a change and then undoing the actions can be very important in today’s world where downtime must often be avoided.

However, even meta data changes to objects like views and stored procedures can be problematic if there are multiple changes made over and over to the same object.

This might seem to indicate that state changes are preferred, but that’s not always the case. Let’s look at a couple of potential database enhancements. Suppose we decide to rename a table. Let’s change the name of the table “Status” to “StatusLookup”. This is a more descriptive name and a common type of refactoring we might do in a C# method, but one that can be problematic for databases.

In this case, we if run a state comparison, we would typically see the view from the SSDT comparison.

(slide)

As you can see, the comparison detects this rename as a drop of one table and an add of another. The script to perform this action would look like this:

(slide)

This is not what we’d like to have happen if there is any data in the original table. Instead, what we would prefer is that the same rename operation that the developer executed is what is placed in the script. In order to do this in a state based database deployment, we would have to edit this script and include custom code that executes the sp\_rename function.

A similar issue occurs with other database operations such as a column split. If we create new columns, we must also include DML code, Data manipulation code, that performs the data movement. This isn’t something a state based comparison tool will generate for us. In fact, there are a whole domain of problems that state based comparison tools cannot handle since there is no way to automatically determine how the objects actually changed.

Since we must maintain state for our table objects, this means custom coding must be written, tracked, and injected into the deployment process. This is a manual process that invites mistakes, something we seek to avoid in a DevOps process.

This does not mean that we cannot use a state based approach, but we must be aware of the places where state based comparisons require additional work. Fortunately there are some products and solutions that