Working with migration scripts

What are migration scripts?

When you deploy changes committed to version control, the SQL Compare engine generates a deployment script to update the target database. You can use a migration script to add custom SQL to a specific point in this deployment script.

Migration scripts are necessary to avoid data loss when making certain schema changes. To achieve this, the migration script intervenes to make data changes occur at the right point of the deployment.

In most cases, you only need to write SQL for the data changes in the migration script. Schema changes are committed separately and deployed as normal.

To learn more, see the examples on this page.

Creating a migration script

To create a new migration script:

1. From the **Object Explorer**, select the database you want to add a migration script to.
2. From the toolbar, select **SQL Source Control**.
   The SQL Source Control window opens.
3. Go to the **Migrations** tab.
4. Select the type of migration script, depending on your development process and the changes you're making:

   ▾ Start from a blank script

   This creates a blank migration script. Select this option if you've already prepared your schema for the migration and committed those changes.

   Example: splitting a column

   1. Schema change
   Create two new columns

   Commit

   2. Migration script
   Split and copy data to new columns

   Commit

   3. Schema change
   Drop the original column

   Commit

   Use the migration script to make any required data changes, then save and commit the script. If you need to make schema changes, commit them separately.

   Schema changes and migration scripts will be deployed in the order they're committed.

   ▾ Include uncommitted schema changes

   Select this option if you've already made all required schema changes for the migration, and haven't yet committed them.

   SQL Source Control generates a migration script which includes DDL changes for the selected objects. Add data changes to the script, then save and commit.

Example: splitting a column

1. Schema change
Create two new columns

-

2. Schema change
Drop original column

-

3. Migration script
Add data changes

Commit

The migration script replaces the section of the deployment script responsible for the selected schema changes.

Changes to selected objects **must occur in the migration script**. Adding or removing DDL changes from the generated script will affect the deployment and might cause data loss.

5. In the **Name** field, enter a name for the script.
6. In the editor window, write SQL to make the required changes.
7. Click **Save & Close**.
8. Commit the changes to version control.

> Always commit a new migration script immediately after saving it. Making changes to your database schema between saving and committing migration scripts can cause errors during deployment.

When you deploy this revision from version control, or use **Get latest** in SQL Source Control on another machine, the migration script will run as part of the deployment. For more information, see How migration scripts work.

Editing migration scripts

You can edit or delete existing migration scripts from the **Migrations** tab in SQL Source Control:

1. From the **Object Explorer**, select a database with migration scripts.
2. From the toolbar, select **SQL Source Control**.
   The SQL Source Control window opens.
3. Go to the **Migrations** tab.
4. Expand **Existing migration scripts.**
   Migration scripts on the remote repository are listed.
5. In the **Actions** column, click **View / Edit** next to a migration script.
6. Edit the script to make the required changes.
   - Guidelines for editing migration scripts
     - Don't create new object dependencies.This is likely to cause errors during deployment.
     - Don't add/remove DDL changes. This might create an invalid state in version control.
     - If you edit the syntax of DDL changes, the resulting schema must stay the same.
7. Click **Save & Close**.
8. Go to the **Commit changes** tab and commit the updated migration script.

Once committed, the updated migration script is used in all future deployments.

Deploying with migration scripts

We recommend using SQL Compare to deploy changes to production, as you have the opportunity to review the deployment script before it's deployed. It is possible to use the **Get latest** function in SQL Source Control to deploy these changes, however we don't recommend linking your production database directly to source control.

Dependencies

When you create a migration script that includes uncommitted schema changes, SQL Source Control automatically includes any dependencies. Deselecting any of these dependencies during the deployment stage will cause the deployment to fail.

Examples

Starting with a blank script

Splitting a column

From a schema point of view, when you split a column you actually create two new columns and drop the original one. If you deploy this change, any data in the dropped column will be lost.

To avoid this, you can write a migration script to copy the data to the new columns *before* drop ping the original column. For a walkthrough of this process, see Splitting a column without data loss.

Example: splitting a column

1. Schema change
Create two new columns.

Commit

2. Migration script
Split and copy data to new columns.

Commit

3. Schema change
Drop the original column.

Commit

Merging columns

From a schema point of view, when you merge a column, you create a new column and drop the original columns. If you deploy this change, any data in the dropped columns will be lost.

To avoid this, you can write a migration script to copy the data from the original columns to the new column *before* dropping the original columns.

Example: merging columns

1. Schema change
Create the new column.

Commit

2. Migration script
Copy data to the new column.

Commit

3. Schema change
Drop the original columns.

Commit

Splitting or merging tables

You can follow the same steps for splitting or merging a tables as you would for splitting or merging columns. To avoid data loss, break the change into multiple commits, and write a migration script to copy the data:

Splitting a table

Commit 1	Commit 2	Commit 3
Create the new table.	Migration script to copy the data to the new table.	Drop the columns in the original table.

Merging tables

Commit 1	Commit 2	Commit 3
Create the new table.	Migration script to copy data from the old tables to the new table.	Drop the original tables.

Adding a NOT NULL constraint to a column

If you add a `NOT NULL` constraint to a column in a table that contains `NULL` entries, without a default value, the deployment will fail.

Rather than adding a default value, you can write a migration script to update all the existing `NULL` entries with a `NOT NULL` value *before* adding the `NOT NULL` constraint to the column:

Example: adding a NOT NULL constraint to a column

1. Migration script
Update `NULL` entries with `NOT NULL` values.

Commit

2. Schema change
Add the `NOT NULL` constraint to the column.

Commit

Changing the data type or size of a column

When you change a column's data type, data might be lost. For example, if the data type you change it to doesn't accommodate some of the rows, data will be truncated during deployment.

To avoid this, you can create a migration script to appropriately modify rows that would otherwise be truncated.

Replacing uncommitted schema changes

Renaming a table

When you rename a table in Management Studio, SQL Source Control interprets this as dropping and recreating the table. If the table contains data, the data will be lost.

To avoid this, you can rename the table using the Object Explorer, then select that uncommitted change on the migrations tab. Generate a migration script for this change and replace the `DROP` and `CREATE` statements in the generated script with the `sp_rename` stored procedure.