The Kentico continuous integration solution handles serialization of data of most objects from the database into XML files on the file system. However, not all database changes are supported by it. Changes to database indexes, functions, stored procedures, custom views, and so on are all unsupported by continuous integration. To keep the main database consistent across your development instances, these changes need to be handled manually. We recommend using migration scripts to synchronize database changes not managed by the Kentico continuous integration solution.

Synchronizing database changes using migration scripts

A migration script allows you to alter a database by changing all or part of a database. This alteration can be as simple as adding or removing a column to a table, or a complex refactoring task such as splitting tables or changing column properties in a way that could affect the data it stores.

The rest of this page assumes you are using a development environment utilizing the Kentico continuous integration solution. See Setting up continuous integration for more information.

Utilizing migration scripts in a continuous integration workflow

The Kentico continuous integration solution supports the use of migration scripts. The ~**CMS\App_Data\CIRepository** folder contains, in addition to serialized database object data, the following files:

- **@migrations** – folder for storing all created migrations connected to a particular repository. Since all migration scripts for a repository are stored in this folder, their file names need to be unique.
- **Before.txt** – holds the file names of all migrations applied before the *CIRepository* folder is restored to the database. Insert only the migration file names without the .sql suffix. Each file name must be inserted on a **separate line**.
- **After.txt** – holds the file names of all migrations applied after the *CIRepository* folder is restored to the database. Insert only the migration file names without the .sql suffix. Each file name must be inserted on a **separate line**.

Additionally, the database of a Kentico instance contains a **CI_Migration** table, which stores the file names of migrations already executed on the database.

When you need to create or modify objects not supported by continuous integration (for example table indexes, functions, or stored procedures) and you want to share these changes with your team members or commit them to a source control system (for example Team Foundation Server or Git), you need to:

1. Write a migration script that applies the required change to the database. Refer to the Example - Creating migration scripts to synchronize database changes section for an example of the process.
2. Reference the migration script in either the *Before.txt* or *After.txt* file.

   If you are not sure when a migration should be executed, see the Deciding when to execute a migration section for specific scenarios and examples.

3. Execute the migration script alongside the continuous integration repository restore on databases you want to keep synchronized.

You can download a sample PowerShell script automating the process here: RunRestore.ps1

Restoring the continuous integration repository via RunRestore.ps1

If you use the *RunRestore.ps1* script, the database restore process consists of these steps:

1. Call the *RunRestore.ps1* script from a command-line interface. Use the full path to a Kentico instance's CMS folder as the argument.
   - For example: .*RunRestore.ps1 C:\inetpub\wwwroot\Kentico11\CMS*
2. The *RunRestore.ps1* script:
   - Retrieves the *CMSConnectionString* from the given project's web.config file.
   - Creates an *App_Offline.htm* file in the directory, stopping the instance.
   - Iterates through migrations referenced in *Before.txt* and applies them to the target database if necessary. The file names of the applied migrations are recorded in the *CI_Migration* table.

- Executes "*ContinuousIntegration.exe -r -s*", deserializing the objects stored in the project's *CIRepository* folder and creates, overwrites or removes corresponding data in the database (specified by the connection string in the given project's web.config file)
- Iterates through migrations referenced in *After.txt* and applies them to the target database if necessary. The file names of the applied migrations are recorded in the *CI_Migration* table.
- Removes the *App_Offline.htm* file from the directory, bringing the instance back online.

The targeted project's database now contains the *CIRepository* data and modifications applied via migration scripts.

Deciding when to execute a migration

After creating a migration script, you need to decide whether it is going to be applied before or after the continuous integration files are restored.

The *Before.txt* file should only contain migrations that manipulate the database before the continuous integration files are restored. These migrations can, for example:

- Drop indexes before their tables are deleted by the restore process.
- Delete foreign key constraints before their tables are deleted by the restore process.

The *After.txt* file should only contain migrations that manipulate the restored database. These migrations can, for example:

- Create new indexes.
- Create new views.
- Add foreign key constraints.
- Add or modify stored procedures.

Important: If you have multiple migrations, carefully consider the order in which they should be executed. The migration scripts in both the *Before.txt* and *After.txt* files are executed from the first to the last line.

Example - Creating migration scripts to synchronize database changes

If you need to create a new index on, for example, a column **OfficeName** in a custom module class table **CompanyOffices** and want this change to be reflected in the repository, you need to:

1. Create a new migration script (for example, **AddOfficeNameIndex.sql**):

   AddOfficeNameIndex.sql

   ```
   CREATE INDEX IX_CompanyOffices_OfficeName ON CompanyOffices(OfficeName);
   ```

2. Add the migration script to the *@migrations* folder.
3. Add the migration name to a new line in the *After.txt* file (the index should only be created after the database is updated).

   After.txt

   ```
   ...
   AddOfficeNameIndex
   ```

4. Commit your changes to your source control repository.

When the repository containing your changes is restored on another database, the **AddOfficeNameIndex.sql** migration is automatically applied (creating the *IX_CompanyOffices_OfficeName* index) during the restore process (as part of the migrations executed after the *CIRepository* folder is restored). The migration is then added to the *CI_Migration* table, marked as applied, and will not be executed again on the target database.

Retrieving the latest version of your project and updating the database

To update your local solution and database to the latest version:

1. Get the latest version of your Kentico solution and the **CIRepository** folder from your source control system.
2. Rebuild the solution in Visual Studio.
3. Launch a command-line interface.
4. Execute the **RunRestore.ps1** script using the full path to your Kentico project's **~\CMS** folder location as the argument (for example: *.\RunRestore.ps1 C:\inetpub\wwwroot\Kentico11\CMS*).

This restores the *CIRepository* folder to the targeted database, executing all unapplied migrations specified in the *Before.txt* and *After.txt* files.

Rolling back database changes

When rolling back your changes, it is important to realize that migrations you have created and committed to source control may have already been applied to other databases. Therefore, removing your migrations from the *@migrations* folder and *Before.txt* or *After.txt* is **not enough**. You also need to a write a new migration that returns the structure of the database where your previous migration was already applied to their original state.

Example - Rolling back changes

The rollback process is demonstrated here by extending the example above. To revert the changes made by the **AddOfficeNameIndex.sql** migration, you need to:

1. Delete the migration script **AddOfficeNameIndex.sql** from the *@migrations* folder.
2. Remove the migration name **AddOfficeNameIndex** from the list of migrations in the *After.txt* file.
3. Create a new migration script (for example, **RemoveOfficeNameIndex.sql**) that removes the index (if it exists):

RemoveOfficeNameIndex.sql

```
IF EXISTS(SELECT * FROM sys.indexes WHERE name = 'IX_CompanyOffices_OfficeName'
AND object_id = OBJECT_ID('CompanyOffices'))
BEGIN
    DROP INDEX IX_CompanyOffices_OfficeName ON CompanyOffices
END
```

4. Add the migration script to the *@migrations* folder.
5. Add the migration name to a new line in the *Before.txt* file (the index should be dropped before its table is deleted during the restore process):

Before.txt

```
...
RemoveOfficeNameIndex
```

6. Commit your changes to your source control repository.

When the repository containing your changes is restored on another database, the *RemoveOfficeNameIndex.sql* migration is automatically applied (removing the *IX_CompanyOffices_OfficeName* index, if it exists) during the restore process (as part of the migrations executed before the *CIRepository* folder is restored). The migration is then added to the *CI_Migration* table, marked as applied, and will not be executed again on the target database.

Restoring the database to a specific point in time

If you need to revert your database to a specific backup, we recommend restoring the backup to a clean database to avoid possible structural database conflicts which may arise when restoring the database to an older version.

To perform a restore on a clean database:

1. Create a blank database.
2. Get the required version of your Kentico solution and the **CIRepository** folder from your source control system.
3. Update the **CMSConnectionString** in the project's web.config file to point to the new database.
4. Rebuild the solution in Visual Studio.
5. Execute the **RunRestore.ps1** script using the full path to your Kentico project's **~\CMS** folder location as the argument (for example: *.\RunRestore.ps1 C:\inetpub\wwwroot\Kentico11\CMS*).

After the restore finishes, you can continue working with the restored database backup.