**Code First Migrations with an existing database**

**EF4.3 Onwards Only** - The features, APIs, etc. discussed in this page were introduced in Entity Framework 4.1. If you are using an earlier version, some or all of the information does not apply.

**Code First Migrations with an existing database**

This topic covers using Code First Migrations with an existing database, one that wasn’t created by Entity Framework.

It’s assumed that you already have an understanding of Code First Migrations. If not, you should read the [Code First Migrations](http://msdn.microsoft.com/en-us/data/jj591621) topic before continuing.

The following topics are covered on this page:

* [Screencasts](http://msdn.microsoft.com/en-us/data/dn579398#screencasts)
* [Step 1: Create a model](http://msdn.microsoft.com/en-us/data/dn579398#step1)
* [Step 2: Enable Migrations](http://msdn.microsoft.com/en-us/data/dn579398#step2)
* [Step 3: Add an initial migration](http://msdn.microsoft.com/en-us/data/dn579398#step3)
  + [Option One: Use existing schema as a starting point](http://msdn.microsoft.com/en-us/data/dn579398#option1)
  + [Option Two: Use empty database as a starting point](http://msdn.microsoft.com/en-us/data/dn579398#option2)
* [Things to be aware of](http://msdn.microsoft.com/en-us/data/dn579398#aware)
  + [Default/calculated names may not match existing schema](http://msdn.microsoft.com/en-us/data/dn579398#names)
  + [Not all database objects are represented in the model](http://msdn.microsoft.com/en-us/data/dn579398#objects)

**Screencasts**

If you'd rather watch a screencast than read this article, the following two videos cover the same content as this article.

**Video One: "Migrations - Under the Hood"**

This screencast covers how migrations tracks and uses information about the model to detect model changes.

<http://channel9.msdn.com/Blogs/EF/Migrations-Under-the-Hood>

**Video Two: "Migrations - Existing Databases"**

Building on the concepts from the previous video, this screencast covers how to enable and use migrations with an existing database.

<http://channel9.msdn.com/Blogs/EF/Migrations-Existing-Databases>

**Step 1: Create a model**

Your first step will be to create a Code First model that targets your existing database. The [Code First to an Existing Database](http://msdn.microsoft.com/en-us/data/jj200620) topic provides detailed guidance on how to do this.

***Note:****It is important to follow the rest of the steps in this topic before making any changes to your model that would require changes to the database schema. The following steps require the model to be in-sync with the database schema.*

**Step 2: Enable Migrations**

The next step is to enable migrations. You can do this by running the **Enable-Migrations** command in Package Manager Console.

This command will create a folder in your solution called Migrations, and put a single class inside it called Configuration. The Configuration class is where you configure migrations for your application, you can find out more about it in the [Code First Migrations](http://msdn.microsoft.com/en-us/data/jj591621) topic.

**Step 3: Add an initial migration**

Once migrations have been created and applied to the local database you may also want to apply these changes to other databases. For example, your local database may be a test database and you may ultimately want to also apply the changes to a production database and/or other developers test databases. There are two options for this step and the one you should pick depends whether or not the schema of any other databases is empty or currently matches the schema of the local database.

* **Option One: Use existing schema as starting point.** You should use this approach when other databases that migrations will be applied to in the future will have the same schema as your local database currently has. For example, you might use this if your local test database currently matches v1 of your production database and you will later apply these migrations to update your production database to v2.
* **Option Two: Use empty database as starting point.** You should use this approach when other databases that migrations will be applied to in the future are empty (or do not exist yet). For example, you might use this if you started developing your application using a test database but without using migrations and you will later want to create a production database from scratch.

**Option One: Use existing schema as a starting point**

Code First Migrations uses a snapshot of the model stored in the most recent migration to detect changes to the model (you can find detailed information about this in [Code First Migrations in Team Environments](http://msdn.microsoft.com/en-us/data/dn481501)). Since we are going to assume that databases already have the schema of the current model, we will generate an empty (no-op) migration that has the current model as a snapshot.

1. Run the **Add-Migration InitialCreate –IgnoreChanges** command in Package Manager Console. This creates an empty migration with the current model as a snapshot.
2. Run the **Update-Database** command in Package Manager Console. This will apply the InitialCreate migration to the database. Since the actual migration doesn’t contain any changes, it will simply add a row to the \_\_MigrationsHistory table indicating that this migration has already been applied.

**Option Two: Use empty database as a starting point**

In this scenario we need Migrations to be able to create the entire database from scratch – including the tables that are already present in our local database. We’re going to generate an InitialCreate migration that includes logic to create the existing schema. We’ll then make our existing database look like this migration has already been applied.

1. Run the **Add-Migration InitialCreate** command in Package Manager Console. This creates a migration to create the existing schema.
2. Comment out all code in the Up method of the newly created migration. This will allow us to ‘apply’ the migration to the local database without trying to recreate all the tables etc. that already exist.
3. Run the **Update-Database** command in Package Manager Console. This will apply the InitialCreate migration to the database. Since the actual migration doesn’t contain any changes (because we temporarily commented them out), it will simply add a row to the \_\_MigrationsHistory table indicating that this migration has already been applied.
4. Un-comment the code in the Up method. This means that when this migration is applied to future databases, the schema that already existed in the local database will be created by migrations.

**Things to be aware of**

There are a few things you need to be aware of when using Migrations against an existing database.

**Default/calculated names may not match existing schema**

Migrations explicitly specifies names for columns and tables when it scaffolds a migrations. However, there are other database objects that Migrations calculates a default name for when applying the migrations. This includes indexes and foreign key constraints. When targeting an existing schema, these calculated names may not match what actually exists in your database.

Here are some examples of when you need to be aware of this:

**If you used ‘Option One: Use existing schema as a starting point’ from Step 3:**

* If future changes in your model require changing or dropping one of the database objects that is named differently, you will need to modify the scaffolded migration to specify the correct name. The Migrations APIs have an optional Name parameter that allows you to do this.  
    
  For example, your existing schema may have a Post table with a BlogId foreign key column that has an index named IndexFk\_BlogId. However, by default Migrations would expect this index to be named IX\_BlogId. If you make a change to your model that results in dropping this index, you will need to modify the scaffolded DropIndex call to specify the IndexFk\_BlogId name.

**If you used ‘Option Two: Use empty database as a starting point’ from Step 3:**

* Trying to run the Down method of the initial migration (i.e. reverting to an empty database) against your local database may fail because Migrations will try to drop indexes and foreign key constraints using the incorrect names. This will only affect your local database since other databases will be created from scratch using the Up method of the initial migration.  
    
  If you want to downgrade your existing local database to an empty state it is easiest to do this manually, either by dropping the database or dropping all the tables. After this initial downgrade all database objects will be recreated with the default names, so this issue will not present itself again.
* If future changes in your model require changing or dropping one of the database objects that is named differently, this will not work against your existing local database – since the names won’t match the defaults. However, it will work against databases that were created ‘from scratch’ since they will have used the default names chosen by Migrations.  
    
  You could either make these changes manually on your local existing database, or consider having Migrations recreate your database from scratch – as it will on other machines.
* Databases created using the Up method of your initial migration may differ slightly from the local database since the calculated default names for indexes and foreign key constraints will be used. You may also end up with extra indexes as Migrations will create indexes on foreign key columns by default – this may not have been the case in your original local database.

**Not all database objects are represented in the model**

Database objects that are not part of your model will not be handled by Migrations. This can include views, stored procedures, permissions, tables that are not part of your model, additional indexes, etc.

Here are some examples of when you need to be aware of this:

* Regardless of the option you chose in ‘Step 3’, if future changes in your model require changing or dropping these additional objects Migrations will not know to make these changes. For example, if you drop a column that has an additional index on it, Migrations will not know to drop the index. You will need to manually add this to the scaffolded Migration.
* If you used ‘Option Two: Use empty database as a starting point’, these additional objects will not be created by the Up method of your initial migration.  
    
  You can modify the Up and Down methods to take care of these additional objects if you wish. For objects that are not natively supported in the Migrations API – such as views – you can use the [Sql](http://msdn.microsoft.com/en-us/library/system.data.entity.migrations.dbmigration.sql(v=vs.113).aspx) method to run raw SQL to create/drop them.