Exercises

By: Mosh Hamedani

Overriding Code-First Conventions

These exercises are built on top of the exercises in the last section: Building a Model using Code-First Workflow.

4th Iteration

In the last exercise, you generated a SQL script for your DBA to deploy the database. He was not happy about the schema you provided and has requested a few changes:

Videos table

- Name column cannot be NULL and its length should be maximum 255 characters.
- Genre_Id column should be renamed to GenreId, and it cannot be NULL.
- Classification column should be a tinyint.

Genres table

- Name column cannot be NULL and its length should be maximum 255 characters.

Implement the requested changes using Fluent API. Organize your configurations into separates, one class per entity (see the lecture “Organizing Fluent API Configurations” for more details).

Read the following hints before making any changes.

Hint 1: Open the Videos table and note the type of Classification column. The reason this column is an **int** is because in C# enums are integers (by default). To change the type of this column, you need to change the Classification enum as follows:

```
public enum Classification : byte
```

Hint 2: Once you override the default conventions and try to run Update-Database, you’ll encounter the following error:

The object 'DF__Videos__Classifi__1920BF5C' is dependent on column 'Classification'. ALTER TABLE ALTER COLUMN Classification failed because one or more objects access this column.

In situations where you receive errors like this about database objects with some random identifiers (eg DF__Videos__Classifi__1920BF5C), you need to inspect your database in SQL Server Management Studio. Expand **Videos > Constraints** and you'll see the constraint. Right-click it and select **Script Constraint as > CREATE TO**. This way you can see what this constraint does. In this case, it sets 0 as the default value for Classification. This was automatically added by one of your earlier code-first migrations.

The reason this constraint was added is because the Classification property of Video class is not nullable. So, in situations like that, Entity Framework assigns default values to prevent the migration from failing in case there are existing records in the table.

So, to finish this exercise, you need to manually drop this unnecessary constraint. Modify your current migration that includes schema changes and add the following line at the beginning:

```
Sql("ALTER TABLE dbo.Videos DROP CONSTRAINT [constraint-name]");
```

Be sure to replace **[constraint-name]** with the name of constraint in your database.

Then run the migration to make the changes to the Videos table.

Hint 3: If you have a table open in the Table Designer in SQL Server Management Studio and then update your database using code-first migrations, you'll not see the changes, even if you right-click on the table and open it again in the designer. You need to close the existing table design session and re-open your table in the designer.

Once you've made all the requested changes, generate a new script for your DBA.

Exercises

By: Mosh Hamedani

5th Iteration

Vidzy is happy with your job and want a new feature: the ability to tag videos. Each video can have zero or more tags and each tag can be applied to zero or more videos.

Change your model to satisfy this requirement. Unlike the last iteration where you created a separate migration to overwrite code-first conventions, in this iteration, override the default conventions at the same time as adding the new classes.

- The intermediary table for the many-to-many relationship between videos and tags should be called **VideoTags**.
- This table should have two columns: **VideoId** and **TagId**.

Apply the changes and create a new database script for your DBA.