Exercises

By: Mosh Hamedani

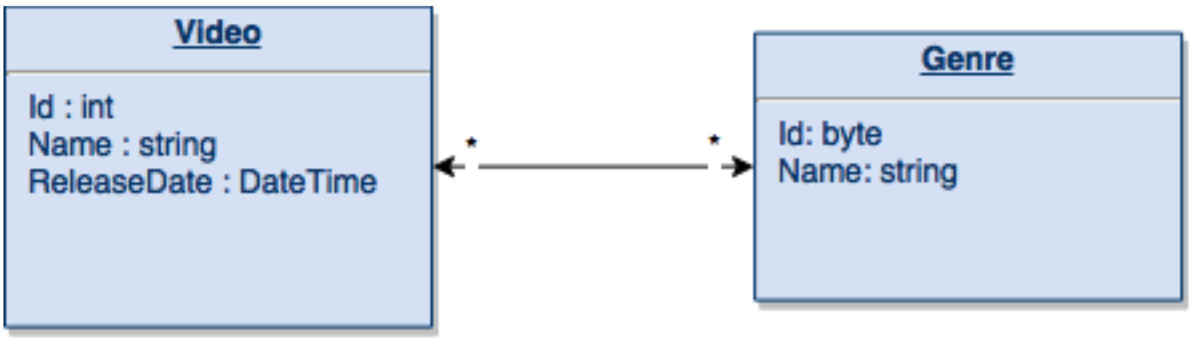
Building a Model using Code First Workflow

Your job is to build an application for a video rental store called Vidzy. For the purpose of this course, you don't need to worry about user interface, so you'll be doing all these exercises in a console application.

1st Iteration

You attempt to build the Vidzy app in an iterative way. In the first iteration, you want to implement the ability to add videos in the database.

Create a new console application and build the following model using the code-first workflow:



Note that there is a many-to-many relationship between **Video** and **Genre**.

Use code-first migrations to generate the database and populate the **Genres** table with some reference data.

Hint: Use two migrations, one for creating the tables, another for populating the Genres table. (reason: if you include your INSERT INTO statements into the InitialModel migration, and then you decide to overwrite it as a result of some recent changes, you'll lose all your INSERT INTO statements.)

Inspect the generated database and its tables.

Exercises

By: Mosh Hamedani

2nd Iteration

You realize that you over-engineered the solution and each Video needs one and only one genre.

Change your model such that each Video has only one Genre.

Use code-first migrations to update the database. Inspect the database and note the changes to your tables.

3rd Iteration

Vidzy tells you that they need to classify their videos into three categories: Silver, Gold and Platinum. You decide to implement this by using an enum: **Classification**. Add a property of type **Classification** to your **Video** class.

Use code-first migrations to update the database. Inspect the database and note the change to the Videos table.

Deployment

You're ready to deploy the application. Your DBA expects you to provide a database script. Run the following command to get a SQL script of all your migrations:

```
Update-Database -Script -SourceMigration:0
```

This command generates the SQL script from the very first migration to the last one. In a real-world scenario, you may want to change the range of migrations included in the SQL script in each deployment. To do this, you can use:

```
Update-Database -Script -SourceMigration:Migr1 -TargetMigration:Migr2
```