# *Web Programming VI (420-H60-HR)*

# *Lab 02a – Entity Framework Review*

Date due:  **August 30, 1 PM (start of next lab)**

**Learning Objectives**

Upon successful completion of this lab exercise, the student will have:

* Use Data First Entity Framework to connect to an existing database
* Use LINQ to query a database
* Use Code First Entity Framework to create a database from classes

Set Up

Create a folder called initials\_H60L02. Both solutions for this lab are to go into this folder.

To Do

**Part A**

1. Create a .NET Core solution in the folder you created above and use the MVC Template. Call the solution PartA.
2. Add the four NuGet Packages needed for Entity Framework. These are:

Microsoft.EntityFrameworkCore.Design;

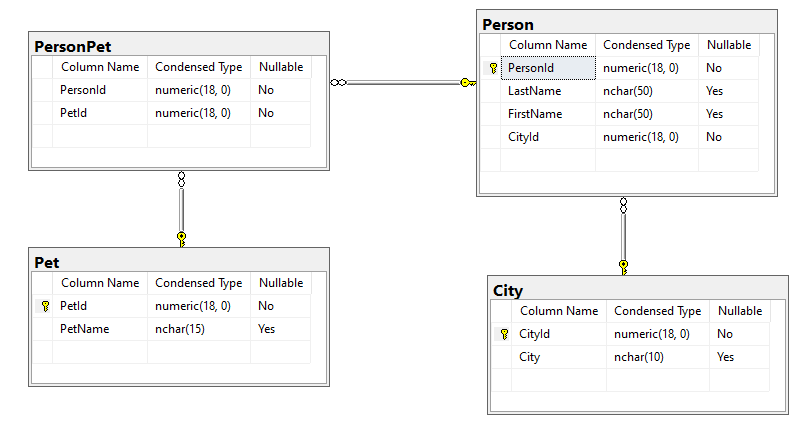
Microsoft.EntityFrameworkCore.SqlServer;

Microsoft.EntityFrameworkCore.Tools;

1. Go to the Package Manager Console and enter the scaffold-dbcontext command to attach to the SQL Server database H60L02EFReview on cssql.cegep-heritage.qc.ca using the account h60 with the password h60. For review…the command looks like this:

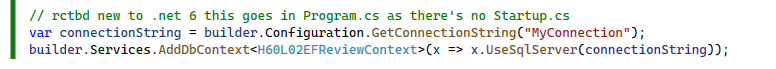


When this command completes successfully you should have a number of classes in your Models folder that have been created. There is a Context class, a Person class, a Pet class, a City Class and a PersonPet class. This is one for each table in the database and one to set the context of the database. The database looks like this:



1. Add the Connection string to appsettings.json with the key-value pair, MyConnection and the connection string you entered in the scaffolding command above.
2. In the Program.cs

~~~~



1. You should remove the string from the DbContext class as well. There’s a warning put in place by the scaffolding, but now that you’ve set it up in the Connection string, you can instead use:

 optionsBuilder.UseSqlServer("Name=ConnectionStrings:MyConnection");

1. Now we are ready to start querying the database using LINQ. It is up to you how many controllers you create for the next section. In the end I want a single page with links to the results of the queries listed below. You can do it in one controller if you like or you can split it up in a controller per database table. (The goal of this lab is to recap Entity Framework).

There must be links from the main page to the controller(s) for each query. You are not to type things into the URL to make this work.

Note: You are welcome to scaffold the views for these queries and to leave the database calls directly in the Controller. This violates what I said about skinny controllers, but this lab is about remember LINQ and not formatting. If you want to practice, then when you are done, move the calls to the model for practice in the upcoming assignment.

1. Using LINQ, set up actions in the controller(s) for the following queries. Each of the actions must call a view to display the results of the query.
   1. A query which lists all the cities in the City table, sorted by City name.
   2. A query which lists all the pets in the pet table that start with the letter P (there are 3).
2. The following queries require joins. Hint: Implement a row count in your tables to show how many rows are being displayed.
   1. List all the first and last name of all people from Gatineau sorted by last name (there are 16)
   2. List the first and last name of all persons ordered by first name who have a pet cat (there are 6)
   3. List all pets (ordered by name) for the person with the last name Jamme (there are 6).
   4. List, sorted by lastname and then firstname, the last name, first name, city and pet for all users. Have the first and last name be displayed in one field. Only list the name and city once and then all the pets for that person. If there are no pets you can display a message No Pets or just leave it blank
   5. List all people (sorted by last name and then first name) as one field that do not have any pets (there are 46)

**Part B**

Note: See [here](https://www.entityframeworktutorial.net/code-first/code-based-migration-in-code-first.aspx) for recap on migrations.

1. Create a new solution in the folder you created in setting up the lab. Call the solution PartB (this will create a folder called PartB).
2. Add the following NuGet Packages: Microsoft.EntityFrameworkCore; Microsoft.EntityFrameworkCore.SqlServer; Microsoft.EntityFrameworkCore.Tools.
3. Open appsettings.json and add a connection string, just like for PartA. This time the connection string is to a database called H60Movies\_*initials* (where *initials* are your initials). The database is on cssql and uses your id and password to connect. (You may have to create a UserId and password as a local login on cssql, with public access to “master”, grant connect sql, grant control server).
4. Create three classes in the Models folder. These will become the tables in your database.
   1. The Movie Class has the attributes MovieId (long), Title (string) and an ICollection<MovieGenre> called MovieGenres (this will give you an error until you define the class in a minute).
   2. The Genre Class has the attributes GenreId (long), GenreName (string) and an ICollection<MovieGenre> called MovieGenres (this will give you an error until you define the class in a minute).
   3. The MovieGenre Class has the attributes MovieGenreId (long), Movie (Movie) and Genre (Genre).
5. Add a new class in the Models folder called MovieContext. Add the Microsoft.EntityFrameworkCore namespace. Declare the class as a derivation of the DBContext class. Declare three properties of type DbSet; one for each of the other classes (tables) you declared. Like: public DbSet<Movie> Movies { get; set; }
6. Open the Program.cs. Add the Models namespace and the EntityFrameworkCore namespace. Add the AddDbContext service similar to what you did in PartA.
7. Now go back to the MovieContext class and add a constructor which has one parameter called options of type DBContextOptions<MovieContext> and simply calls the parent constructor. Like this:  
   public MovieContext(DbContextOptions<MovieContext> options) : base(options) {

Database.EnsureCreated(); // this has to be commented out for migrations to be run

}

1. Now let’s add some initial seed data to the database. At the bottom of the Context class you created (MovieContext) add a protected method which overrides the OnModelCreating method. Like this:   
   

Now we’ll add some Genres by adding commands to the Genre entity like this:  
  
Add 5 Genres in total. Use a similar technique to add 12 Movies as well

1. Open the Package Manager Console and enter the command add-migration initialDB. This should run successfully and create the Database for you. To view it, go to SQL Server Management Studio and open (connect to) cssql.
2. You should have a database with three tables all linked by foreign keys allowing a movie to have many genres and a genre to be for many movies. Open the Movies and Genres tables and make sure the data you added is there. (make sure that the movies are bound to the genre in the associative entity).
3. Now we're going to change the Movie class to add the attribute called Rating which is a decimal number between 0.0 and 100.0.
4. Add seed data to give each of your movies a score.
5. Add a migration for the adding of the field. Update the database using the migration. Use SSMS to make sure the database is correct.

**Marking**

|  |  |
| --- | --- |
| **Web Programming VI (420-H60-HR ) Marks Lab 2 (MVC + EF Review)** | **Out of** |
| **Part A: Data First** |  |
| Scaffold in the classes | 3 |
| Set MyConnection in appsettings and startup | 2 |
| Default page with links | 1 |
| All cities sorted by city name | 2 |
| All pets starting with P (3) | 2 |
| Names of people in Gatineau (16) | 3 |
| Cat Pet (8) | 4 |
| Pets for Jamme (6) | 5 |
| All people, city and pets | 5 |
| All people with no pets (46) | 3 |
| **Part B - Code First** |  |
| Create classes | 4 |
| Create Context | 3 |
| Seed Data | 4 |
| First migration | 2 |
| Add Rating/Second migration | 4 |
| Final DB | 3 |
| Handed in Properly | 4 |
| **Total** | 54 |

**To submit**

When you have completed the lab exercise show me your work and then copy the zip file to Moodle.