### Exercise 5: Creating a Database Using Code Firstv

In this exercise, you will learn how to use the Code First approach to create a database with the tables of the MusicStore application, and how to access its data.

Once the model is generated, you will modify the StoreController to provide the View template with the data taken from the database, instead of using hardcoded values.

**Note:** Add Entity Framework first from Nuget Package.

#### **Task 1 - Working with the Model**

Now that you have already configured the connection to the database, you will link the model with the database tables. In this task, you will create a class that will be linked to the database with Code First. Remember that there is an existent POCO model class that should be modified.

**Note:** If you completed Exercise 1, you will note that this step was performed by a wizard. By doing Code First, you will manually create classes that will be linked to data entities.

1. Open the POCO model class **Genre** from **Models** project folder and include an ID. Use an int property with the name **GenreId**.

(Code Snippet - Models And Data Access - Ex2 Code First Genre)

C#

namespace MvcMusicStore.Models

{

using System.Collections.Generic;

public class Genre

{

**public int GenreId { get; set; }**

public string Name { get; set; }

public string Description { get; set; }

public List<Album> Albums { get; set; }

}

}

**Note:** To work with Code First conventions, the class Genre must have a primary key property that will be automatically detected.

You can read more about Code First Conventions in this [msdn article](http://msdn.microsoft.com/en-us/library/hh161541&#040;v=vs.103&#041;.aspx).

1. Now, open the POCO model class **Album** from **Models** project folder and include the foreign keys, create properties with the names**GenreId** and **ArtistId**. This class already have the **GenreId** for the primary key.

(Code Snippet - Models And Data Access - Ex2 Code First Album)

C#

namespace MvcMusicStore.Models

{

public class Album

{

public int AlbumId { get; set; }

**public int GenreId { get; set; }**

**public int ArtistId { get; set; }**

public string Title { get; set; }

public decimal Price { get; set; }

public string AlbumArtUrl { get; set; }

public virtual Genre Genre { get; set; }

public virtual Artist Artist { get; set; }

}

}

1. Open the POCO model class **Artist** and include the **ArtistId** property.

(Code Snippet - Models And Data Access - Ex2 Code First Artist)

C#

namespace MvcMusicStore.Models

{

public class Artist

{

**public int ArtistId { get; set; }**

public string Name { get; set; }

}

}

1. Right-click the **Models** project folder and select **Add | Class**. Name the file **MusicStoreEntities.cs**. Then, click **Add.**

Adding a new item

Adding a class

1. Open the class you have just created, **MusicStoreEntities.cs**, and include the namespaces **System.Data.Entity** and**System.Data.Entity.Infrastructure**.

C#

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

**using System.Data.Entity;**

**using System.Data.Entity.Infrastructure;**

**...**

1. Replace the class declaration to extend the **DbContext** class: declare a public **DBSet** and override **OnModelCreating** method. After this step you will get a domain class that will link your model with the Entity Framework. In order to do that, replace the class code with the following:

(Code Snippet - Models And Data Access - Ex2 Code First MusicStoreEntities)

C#

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Data.Entity;

using System.Data.Entity.Infrastructure;

namespace MvcMusicStore.Models

{

**public class MusicStoreEntities : DbContext**

**{**

**public DbSet<Genre> Genres { get; set; }**

**public DbSet<Album> Albums { get; set; }**

**public DbSet<Artist> Artists { get; set; }**

**protected override void OnModelCreating(DbModelBuilder modelBuilder)**

**{**

**modelBuilder.Entity<Genre>().ToTable("Genres");**

**modelBuilder.Entity<Album>().ToTable("Albums");**

**modelBuilder.Entity<Artist>().ToTable("Artists");**

**base.OnModelCreating(modelBuilder);**

**}**

**}**

}

**Note:** With Entity Framework **DbContext** and **DBSet** you will be able to query the POCO class Genre. By extending **OnModelCreating**method, you are specifying in the **code** how Genre will be mapped to a database table. You can find more information about DBContext and DBSet in this msdn article: [link](http://msdn.microsoft.com/en-us/library/system.data.entity.dbcontext(v=vs.103).aspx)

#### **Task 2 - Querying the Database**

In this task, you will update the StoreController class so that, instead of using hardcoded data, it will retrieve it from the database.

**Note:** This task is in common with Exercise 1.

If you completed Exercise 1 you will note these steps are the same in both approaches (Database first or Code first). They are different in how the data is linked with the model, but the access to data entities is yet transparent from the controller.

1. Open **Controllers\StoreController.cs** and add the following field to the class to hold an instance of the **MusicStoreEntities** class, named**storeDB**:

(Code Snippet - Models And Data Access - Ex1 storeDB)

C#

public class StoreController : Controller

{

**private MusicStoreEntities storeDB = new MusicStoreEntities();**

...

}

1. The **MusicStoreEntities** class exposes a collection property for each table in the database. Update **Browse** action method to retrieve a Genre with all of the **Albums**.

(Code Snippet - Models And Data Access - Ex2 Store Browse)

C#

public ActionResult Browse(string genre)

{

// Retrieve Genre and its Associated Albums from database

**var genreModel = new Genre**

**{**

**Name = genre,**

**Albums = this.storeDB.Albums.ToList()**

**};**

return this.View(genreModel);

}

**Note:** You are using a capability of .NET called **LINQ** (language-integrated query) to write strongly-typed query expressions against these collections - which will execute code against the database and return objects that you can program against.

For more information about LINQ, please visit the [msdn site](http://msdn.microsoft.com/en-us/library/bb397926(v=vs.110).aspx).

1. Update **Index** action method to retrieve all the genres.

(Code Snippet - Models And Data Access - Ex2 Store Index)

C#

public ActionResult Index()

{

**var genres = this.storeDB.Genres;**

return this.View(genres);

}

1. Update **Index** action method to retrieve all the genres and transform the collection to a list.

(Code Snippet - Models And Data Access - Ex2 Store GenreMenu)

C#

// GET: /Store/GenreMenu

[ChildActionOnly]

public ActionResult GenreMenu()

{

**var genres = this.storeDB.Genres.Take(9).ToList();**

return this.PartialView(genres);

}

#### **Task 1 - Populating Sample Data**

In this task, you will populate the database with sample data when it is intially created using Code-First.

1. Add the **SampleData.cs** file to the **Models** folder. To do that, right-click **Models** folder, point to **Add** and then click **Existing Item**. Browse to **\Source\Assets\Data** and select the **SampleData.cs** file.
2. Open the **Global.asax.cs** file and add the following using statements.

C#

**using MvcMusicStore.Models;**

**using System.Data.Entity;**

1. In the **Application\_Start()** method add the following line to set the database initializer.

(Code Snippet - Models And Data Access - Ex2 Global Asax SetInitializer)

C#

protected void Application\_Start()

{

AreaRegistration.RegisterAllAreas();

WebApiConfig.Register(GlobalConfiguration.Configuration);

FilterConfig.RegisterGlobalFilters(GlobalFilters.Filters);

RouteConfig.RegisterRoutes(RouteTable.Routes);

BundleConfig.RegisterBundles(BundleTable.Bundles);

**Database.SetInitializer(new SampleData());**

}

#### **Task 3 - Configuring the connection to the Database**

Now that you have already added a database to our project, you will write in the **Web.config** file the connection string.

1. Add a connection string at **Web.config**. To do that, open **Web.config** at project root and replace the connection string named DefaultConnection with this line in the **<connectionStrings>** section:

Web.config file location

XML

<configuration>

...

<connectionStrings>

**<add name="MusicStoreEntities" connectionString="data source=(LocalDb)\v11.0;initial catalog=MvcMusicStore;Integrated Security=SSPI;AttachDBFilename=|DataDirectory|\MvcMusicStore.mdf" providerName="System.Data.SqlClient" />**

</connectionStrings>

...

#### **Task 4 - Running the Application**

In this task, you will check that the Store Index page will now display the Genres stored in the database instead of the hardcoded ones. There is no need to change the View template because the **StoreController** is returning the same **StoreIndexViewModel** as before, but this time the data will come from the database.

1. Rebuild the solution and press **F5** to run the Application.
2. The project starts in the Home page. Verify that the menu of **Genres** is no longer a hardcoded list, and the data is directly retrieved from the database.

Browsing Genres from the database

1. Now browse to any genre and verify the albums are populated from database.

Browsing Albums from the database

### Exercise 3: Querying the Database with Parameters

In this exercise, you will learn how to query the database using parameters, and how to use Query Result Shaping, a feature that reduces the number database accesses retrieving data in a more efficient way.

**Note:** For further information on Query Result Shaping, visit the following [msdn article](http://msdn.microsoft.com/en-us/library/bb896272&#040;v=vs.100&#041;.aspx).

#### **Task 1 - Modifying StoreController to Retrieve Albums from Database**

In this task, you will change the **StoreController** class to access the database to retrieve albums from a specific genre.

1. Open the **StoreController** class to change the **Browse** action method. To do this, in the **Solution Explorer**, expand the **Controllers** folder and double-click **StoreController.cs**.
2. Change the **Browse** action method to retrieve albums for a specific genre. To do this, replace the following code:

(Code Snippet - Models And Data Access - Ex3 StoreController BrowseMethod)

C#

public ActionResult Browse(string genre)

{

// Retrieve Genre and its Associated Albums from database

**var genreModel = this.storeDB.Genres.Include("Albums")**

**.Single(g => g.Name == genre);**

return this.View(genreModel);

}

**Note:** To populate a collection of the entity, you need to use the **Include** method to specify you want to retrieve the albums too. You can use the .**Single()** extension in LINQ because in this case only one genre is expected for an album. The **Single()** method takes a Lambda expression as a parameter, which in this case specifies a single Genre object such that its name matches the value defined.

You will take advantage of a feature that allows you to indicate other related entities you want loaded as well when the Genre object is retrieved. This feature is called **Query Result Shaping**, and enables you to reduce the number of times needed to access the database to retrieve information. In this scenario, you will want to pre-fetch the Albums for the Genre you retrieve.

The query includes **Genres.Include("Albums")** to indicate that you want related albums as well. This will result in a more efficient application, since it will retrieve both Genre and Album data in a single database request.

#### **Task 2 - Running the Application**

In this task, you will run the application and retrieve albums of a specific genre from the database.

1. Press **F5** to run the Application.
2. The project starts in the Home page. Change the URL to **/Store/Browse?genre=Pop** to verify that the results are being retrieved from the database.

Browsing /Store/Browse?genre=Pop

#### **Task 3 - Accessing Albums by Id**

In this task, you will repeat the previous procedure to get albums by their Id.

1. Close the browser if needed, to return to Visual Studio. Open the **StoreController** class to change the **Details** action method. To do this, in the **Solution Explorer**, expand the **Controllers** folder and double-click **StoreController.cs**.
2. Change the **Details** action method to retrieve albums details based on their **Id**. To do this, replace the following code:

(Code Snippet - Models And Data Access - Ex3 StoreController DetailsMethod)

C#

// GET: /Store/

public ActionResult Details(int id)

{

**var album = this.storeDB.Albums.Find(id);**

if (album == null)

{

return this.HttpNotFound();

}

return this.View(album);

}

#### **Task 4 - Running the Application**

In this task, you will run the Application in a web browser and obtain album details by their Id.

1. Press **F5** to run the Application.
2. The project starts in the Home page. Change the URL to **/Store/Details/51** or browse the genres and select an album to verify that the results are being retrieved from the database.



Browsing /Store/Details/51

**Note:** Additionally, you can deploy this application to Windows Azure Web Sites following [Appendix B: Publishing an ASP.NET MVC 4 Application using Web Deploy](http://www.asp.net/mvc/overview/older-versions/hands-on-labs/aspnet-mvc-4-models-and-data-access#AppendixB).