* 1. 

Hands-On Lab

ASP.NET MVC Models and Data Access

* 1. Lab version: 1.1.0
  2. Last updated: 1/12/2011

Contents

[Overview 3](#_Toc282594782)

[Exercise 1: Adding a Database 6](#_Toc282594783)

[Task 1 – Adding a Database 6](#_Toc282594784)

[Task 2 – Creating a Data Model 13](#_Toc282594785)

[Task 3 – Building the Application 18](#_Toc282594786)

[Task 4 – Querying the Database 20](#_Toc282594787)

[Task 5 – Running the Application 21](#_Toc282594788)

[Exercise 2: Adding a Database Using Code First 22](#_Toc282594789)

[Task 1 – Adding a Database 23](#_Toc282594790)

[Task 2 – Including Code First in the Solution and Connecting the Database 30](#_Toc282594791)

[Task 3 – Working with the Model 32](#_Toc282594792)

[Task 4 – Querying the Database 36](#_Toc282594793)

[Task 5 – Running the Application 38](#_Toc282594794)

[Exercise 3: Querying the Database with Parameters 39](#_Toc282594795)

[Task 1 – Modifying StoreController to Retrieve Albums from Database 39](#_Toc282594796)

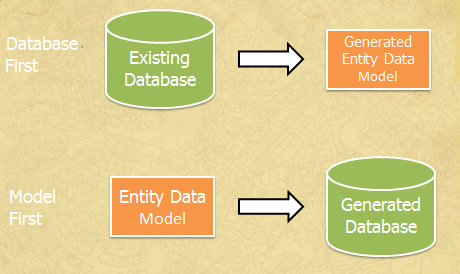
[Task 2 – Running the Application 40](#_Toc282594797)

[Task 3 – Accessing Albums by Id 41](#_Toc282594798)

[Task 4 – Running the Application 42](#_Toc282594799)

[Summary 42](#_Toc282594800)

Overview

* 1. **Note:** This Hands-on Lab assumes you have basic knowledge of **ASP.NET MVC**. If you have not used **ASP.NET MVC** before, we recommend you to go over **ASP.NET MVC Fundamentals** Hand-on Lab.
  2. In **ASP.NET MVC Fundamentals** Hand-on Lab, you have been passing hard-coded data from the Controllers to the View templates. But in order to build a real web application you might want to use a real database.
  3. This Hands-on Lab will show you how to use the free SQL Server Express as a database engine in order to store and retrieve the data needed for the Music Store application. To accomplish that, you will start with an already created database from which you will create the Entity Data Model for the application. Through this lab, you will meet the **Database First** approach and the **Code First** Approach as well.
  4. However, you could use a **Model First** approach, by creating the same model using the tools and then generating a database from it.
  5. 
  6. Figure
  7. Database First vs. Model First
  8. After generating the Model, you will make the proper adjustments in the StoreController to provide the Store Views with the data taken from the database instead of hard-coded one. You will not need to make any change to the View templates because the StoreController will be returning to the View templates the same ViewModels as before, although this time the data will come from the database.

## The Code First Approach

* 1. The Code First approach allows us to define the model from the code without generating classes that are generally coupled with the framework.
  2. In code first, Model objects are defined with POCOs , “Plain Old CLR Objects”. POCOs are defined as simple plain classes that have no inheritance and do not implement interfaces. We can automatically generate the database from them, or we can use an existing database and generate the class mapping from the code.

The benefit of using this approach is that the Model remains independent from the persistence framework (in this case, Entity Framework), as the POCOs classes are not coupled with the mapping framework.

* 1. **Note:** Although that this Hands-on Lab will cover the use of the free SQL Server Express, the code will also work with the full version of SQL Server.
  2. This Lab is based on ASP.NET MVC 3.
  3. If you wish to explore the whole Music Store tutorial application you can find it in <http://mvcmusicstore.codeplex.com/>.



**System Requirements**

You must have the following items to complete this lab:

* 1. ASP.NET and ASP.NET MVC 3
  2. Visual Studio 2010 Express
  3. SQL Server Database (Express edition or above)
     1. **Note**: You can install the previous system requirements by using the Web Platform Installer 3.0: <http://go.microsoft.com/fwlink/?LinkID=194638>.

**Setup**

***Installing Code Snippets***

For convenience, much of the code you will be managing along this lab is available as Visual Studio code snippets. To install the code snippets run **.\Source\Assets\CodeSnippets.vsi** file.

# Exercises

* 1. This Hands-On Lab is comprised by the following exercises:
  2. Exercise 1: Adding a Database
  3. Exercise 2: Adding a Database using Code First
  4. Exercise 3: Querying the Database with Parameters
  5. Estimated time to complete this lab: **35 minutes**.
  6. **Note:** Each exercise is accompanied by an **End** folder containing the resulting solution you should obtain after completing the exercises. You can use this solution as a guide if you need additional help working through the exercises.

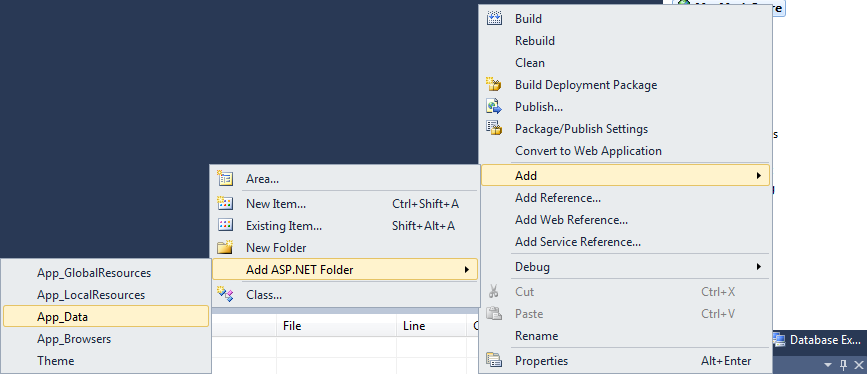
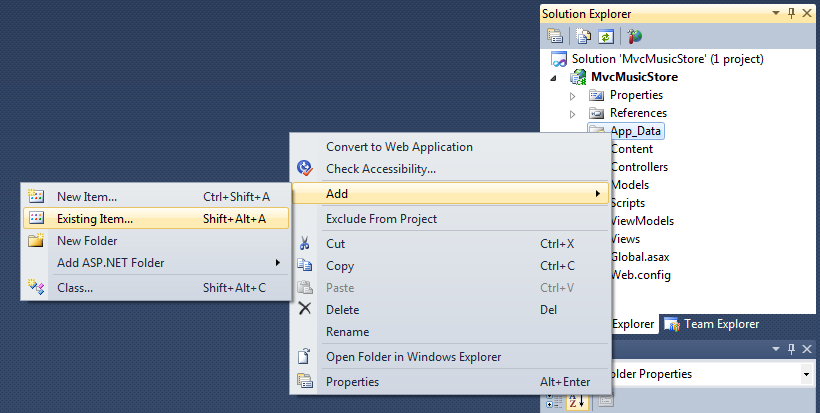
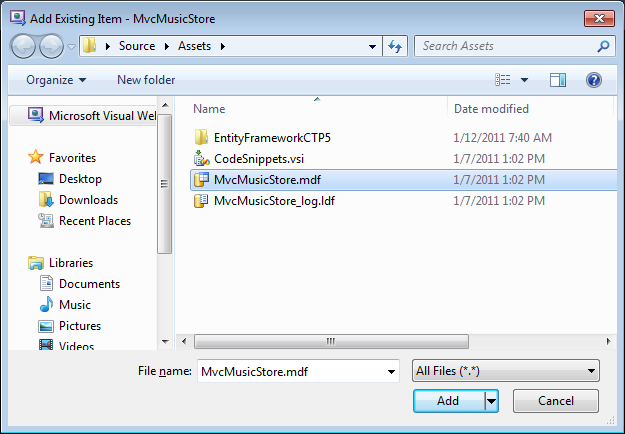
# Next Step

* 1. Exercise 1: Adding a Database

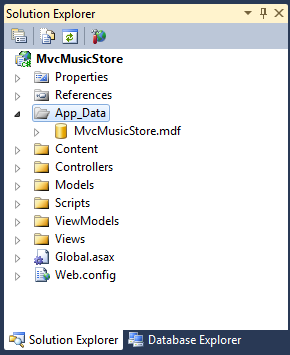
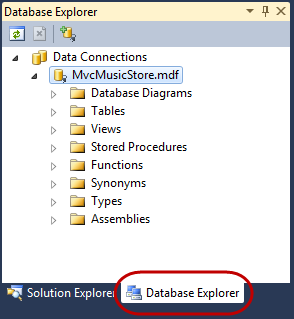
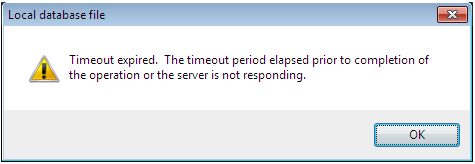
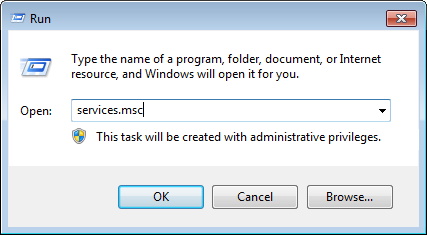
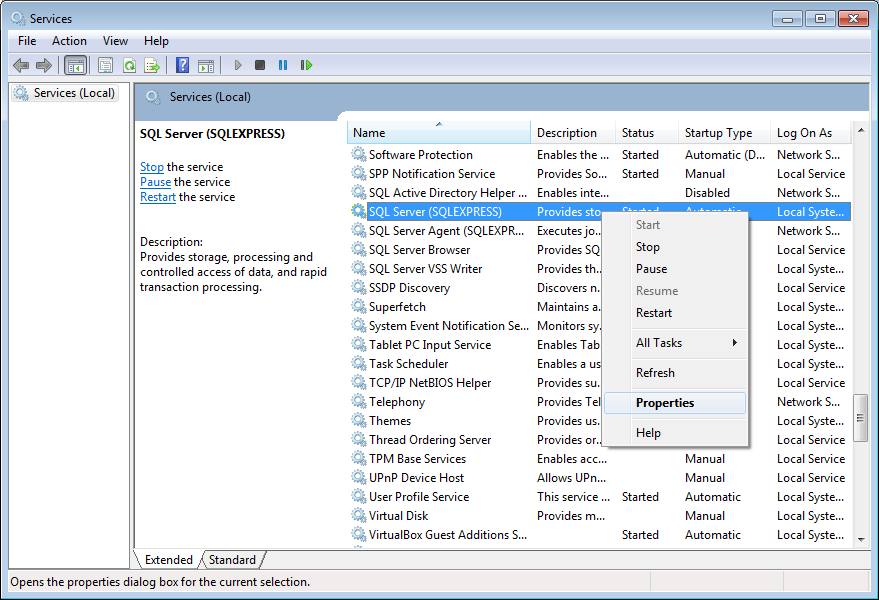
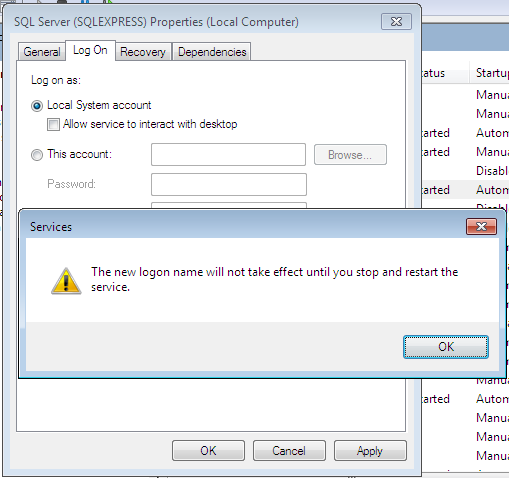
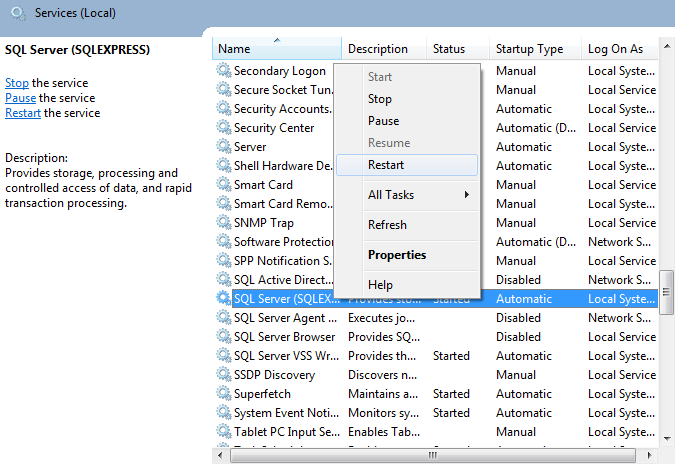
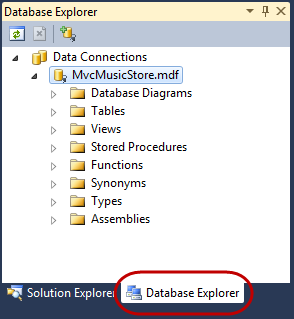
Exercise 1: Adding a Database

* 1. In this exercise, you will learn how to add a database with the tables of the MusicStore application to the solution in order to consume its data. Once adding the database and generating the Model it will represent, you will make the proper adjustments in the StoreController class to provide the View template with the data taken from the database instead of hard-coded one.

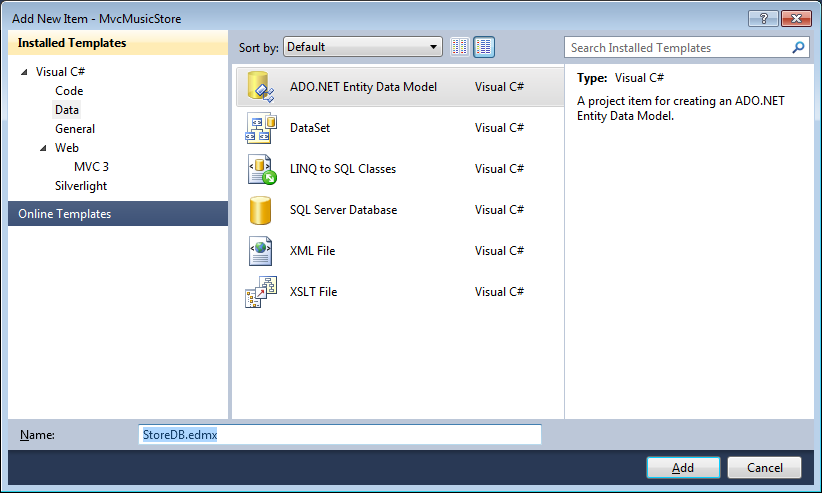
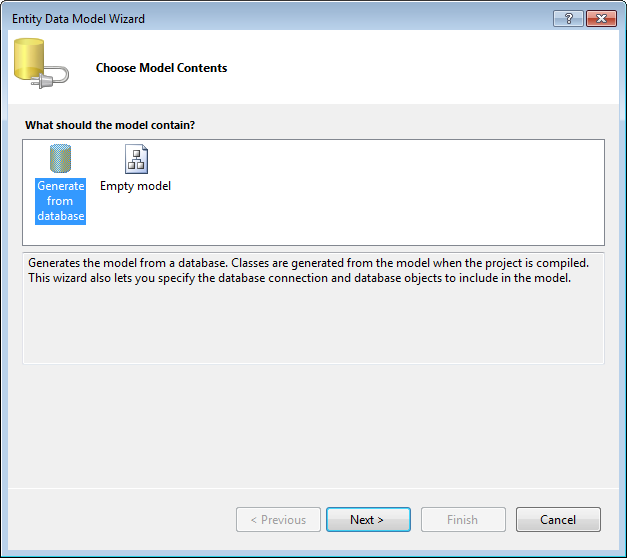
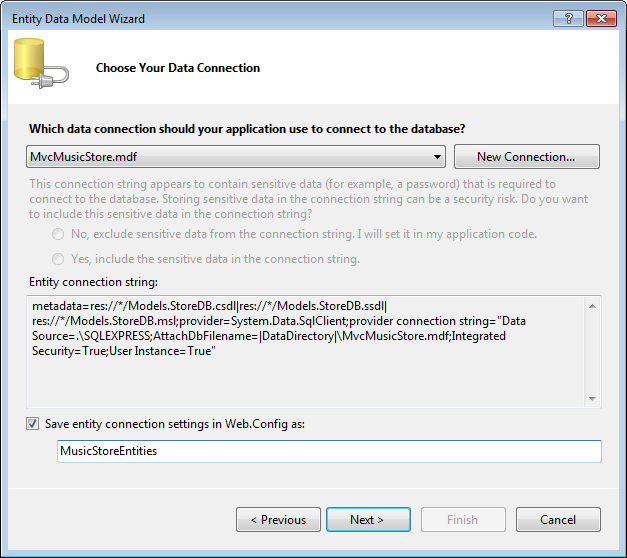
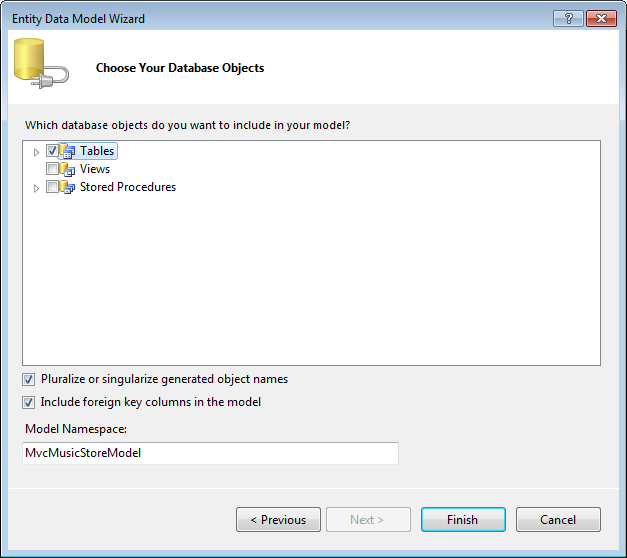
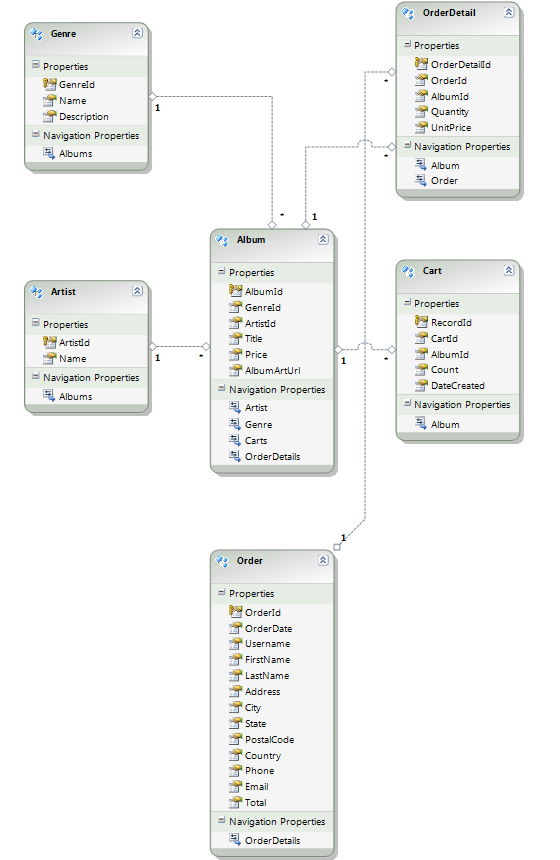
Task 1 – Adding a Database

* 1. In this task, you will add an already created database with the main tables of the MusicStore application to the solution.
  2. Start Microsoft Visual Web Developer 2010 Express from **Start** | **All Programs** | **Microsoft Visual Studio 2010 Express** | **Microsoft Visual Web Developer 2010 Express**.
  3. In the **File** menu, choose **Open Project**. In the Open Project dialog, browse to Source\Ex01-AddingADatabaseDBFirst\Begin, select **MvcMusicStore.sln** and click **Open**.
  4. Add an **App\_Data** folder to the project to hold the SQL Server Express database files. **App\_Data** is a special folder in ASP.NET which already has the correct security access permissions for database access. To add the folder, right-click **MvcMusicStore** project, point to **Add** then to **Add ASP.NET Folder** and finally click **App\_Data**.
     1. 
     2. Figure
     3. Adding an App\_Data folder
  5. Add **MvcMusicStore** database file. In this hands-On Lab, you will use an already created database called **MvcMusicStore.mdf**. To do that, right-click the new **App\_Data** folder, point to **Add** and then click **Existing Item**. Browse to **\Source\Assets\** and select the **MvcMusicStore.mdf** file.
     1. 
     2. Figure
     3. Adding an Existing Item
     4. 
     5. Figure
     6. MvcMusicStore.mdf database file

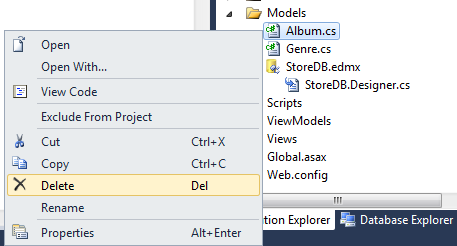
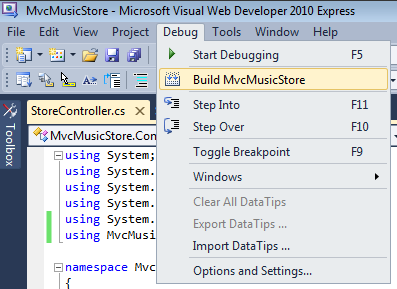
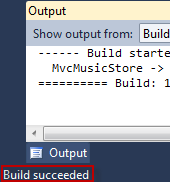
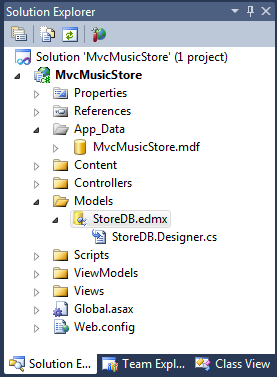
The database has been added to the project. Even when the database is located inside the solution, you can query and update it as it was hosted in a different database server.

* + 1. 
    2. Figure
    3. MvcMusicStore database in Solution Explorer
  1. Verify the connection to the database. To do this, open the **Database Explorer** (CTRL+ALT+S), and then double-click the **MvcMusicStore.mdf**. The connection is established.
     1. 
     2. Figure
     3. Connecting to MvcMusicStore.mdf
     4. **Note:** If you getanerror like the following, please follow the steps below.
     5. 
     6. 1. Open the Windows Services console. To do that, open the Run command from **Start** | **All Programs** | **Accessories** | **Run**, type **services.msc** and then click **OK.**
     7. 
     8. **Figure 7**
     9. *Running services.msc*
     10. 2. Right-click the **SQL Server (SQLEXPRESS)** service and select **Properties**.
     11. 
     12. **Figure 8**
     13. *SQL Server (SQLEXPRESS) service*
     14. 3. Open the **Log On** tab, select **Local System account** as the account to log on with and click **OK**. Accept the dialog by clicking **OK** again.
     15. 
     16. **Figure 9**
     17. *Changing the log on account*
     18. 4. Restart the SQL Server (SQLEXPRESS) service.
     19. 
     20. **Figure 10**
     21. *Restarting SQL Server (SQLEXPRESS) service*
     22. 5. Once the service is restarted, close the **Services** console and verify the connection to the database. To do this, select the **Database Explorer**, and then double-click the **MvcMusicStore.mdf**. The connection is established.
     23. 
     24. **Figure 11**
     25. *Connecting to MvcMusicStore.mdf*

Task 2 – Creating a Data Model

* 1. In this task, you will create a data model to interact with the database added in the previous task.
  2. Create a data model that will represent the added database. To do this, in Solution Explorer right-click the **Models** folder, point to **Add** and then click **New Item**. In the **Add New Item** dialog, select the **Data** template and then the **ADO.NET Entity Data Model** item. Change the data model name to **StoreDB.edmx** and click **Add**.
     1. 
     2. Figure
     3. Adding the StoreDB ADO.NET Entity Data Model
  3. The **Entity Data Model Wizard** appears. This wizard will guide you through the creation of the model layer. Since the model should be created based on the existing database added in the last task, select **Generate from database** and click **Next**.
     1. 
     2. Figure
     3. Choosing the model content
  4. Since you are generating a model from a database, you will need to specify which database to use. The wizard detects the database in the App\_Data folder, so it fills in the correct connection information for that database. The generated class will have the same name as the entity connection string, so change it to **MusicStoreEntities** and click **Next**.
     1. 
     2. Figure
     3. Choosing the data connection
  5. Choose the database objects to use. Since the Entity Model will use just the database’s tables, check the **Tables** checkbox and make sure that the **Include foreign key columns in the model** checkbox is also checked. Change the Model Namespace to **MvcMusicStoreModel** and click **Finish**.
     1. 
     2. Figure
     3. Choosing the database objects
  6. An entity diagram for the database appears. A separate class that maps to each table within the database will be created. For example, the **Albums** table will be represented by an **Album** class with each column in the table mapping to a property on the class. This will allow you to query and work with objects that represent rows within the database. You will see other classes that you might not use in the Hands-on Lab but belong to the Music Store application.
     1. 
     2. Figure
     3. Entity diagram

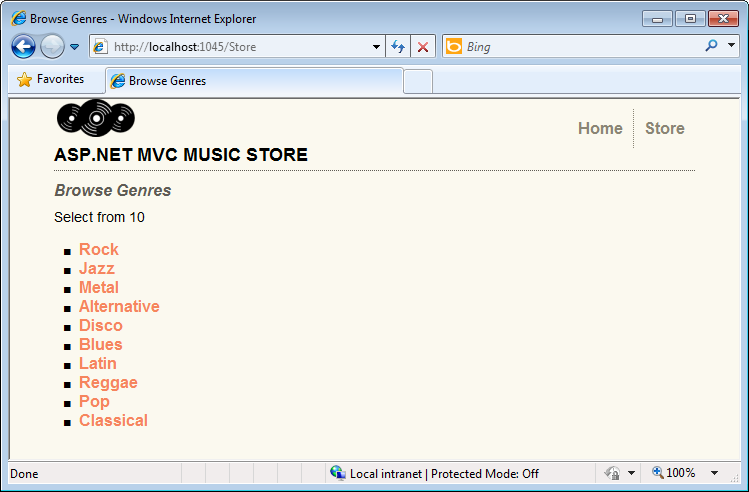
Task 3 – Building the Application

* 1. In this task, you will check that although you have removed the **Album** and **Genre** model classes, the project gets built successfully, by using the classes in the data model.
  2. Delete the placeholder **Album** and **Genre** classes. To do this, in the **Solution Explorer**, expand the **Models** folder, right-click **Album** and select **Delete**. Repeat this procedure with the **Genre** class.
     1. 
     2. Figure
     3. Deleting placeholder classes
  3. Build the project by selecting the **Debug** menu item and then **Build MvcMusicStore**.
     1. 
     2. Figure
     3. Building the project
  4. The project builds successfully. Why does still work? It works because the database tables have fields which include the properties you were using in the earlier **Album** and **Genre** classes manually removed. Data model classes are a drop-in replacement.
     1. 
     2. Figure
     3. Builds succeeded
  5. While the designer displays the entities in a diagram format, they are really C# classes. Expand the **StoreDB.edmx** node in the Solution Explorer, and you will see a file called **StoreDB.Designer.cs.**
     1. 
     2. Figure
     3. StoreDB.Designer.cs file

Task 4 – Querying the Database

* 1. In this task, you will update the StoreController class so that instead of using hard-coded data, it queries the database to retrieve all its information.
  2. Open **Controllers\StoreController.cs** and add the following field to the class to hold an instance of the **MusicStoreEntities** class, named **storeDB**:
     1. *(Code Snippet – ASP.NET MVC Models and Data Access – Ex1 storeDB – CSharp)*
     2. C#
     3. public class StoreController : Controller
     4. {
     5. **MusicStoreEntities storeDB = new MusicStoreEntities();**
  3. The **MusicStoreEntities** class exposes a collection property for each table in the database. Update **StoreController**’s **Index** action method to retrieve all **Genre** names in the database. This was done previously by hard-coding string data. Now you can instead write a LINQ query expression like below which retrieves the **Name** property of each Genre within the database:
     1. *(Code Snippet – ASP.NET MVC Models and Data Access – Ex1 Store Index – CSharp)*
     2. C#
     3. //
     4. // GET: /Store/
     5. public ActionResult Index()
     6. {
     7. **// Retrieve the list of genres**
     8. **var genres = from genre in storeDB.Genres**
     9. **select genre.Name;**
     10. // Create your view model
     11. }
     12. **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.
     13. For more information about LINQ, please visit the [msdn site](http://msdn.microsoft.com/en-us/netframework/aa904594.aspx).
  4. Transform the collection of genres to a list. To do this, replace the following code:
     1. *(Code Snippet – ASP.NET MVC Models and Data Access – Ex1 Store Index ToList – CSharp)*
     2. C#
     3. public ActionResult Index()
     4. {
     5. // Retrieve the list of genres
     6. var genres = from genre in storeDB.Genres
     7. select genre.Name;
     8. // Create your view model
     9. var viewModel = new StoreIndexViewModel
     10. {
     11. **Genres = genres.ToList(),**
     12. **NumberOfGenres = genres.Count()**
     13. };
     14. return View(viewModel);
     15. }

Task 5 – Running the Application

* 1. In this task, you will check that the Store Index page will now display the Genres stored in the database instead of the hard-coded ones. There is no need of changing the View template because the **StoreController** is returning the same **StoreIndexViewModel** as before, although this time the data will come from the database.
  2. Press **F5** to run the Application.
  3. The project starts in the Home page. Change the URL to **/Store** to verify that the list of **Genres** is no longer the hard-coded list, else the ones taken from the database.
     1. ****
     2. Figure
     3. Browsing Genres from the database

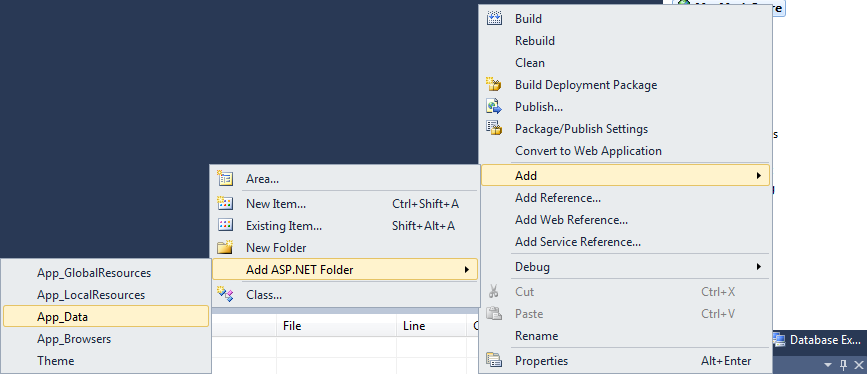
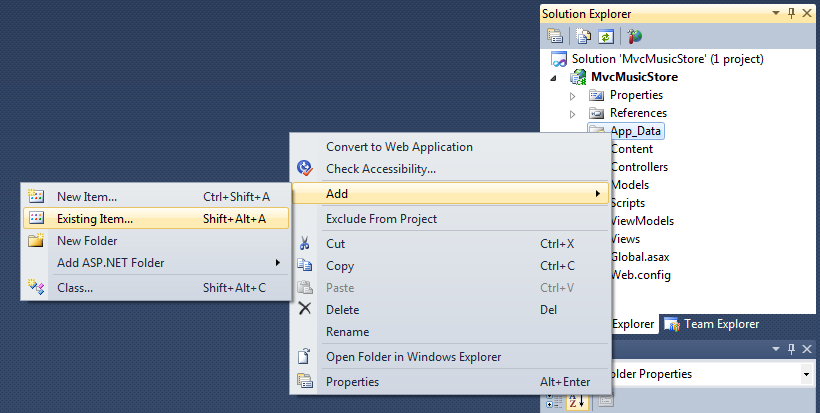
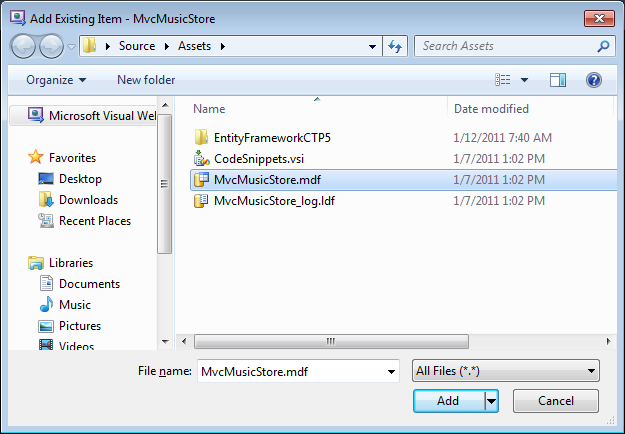
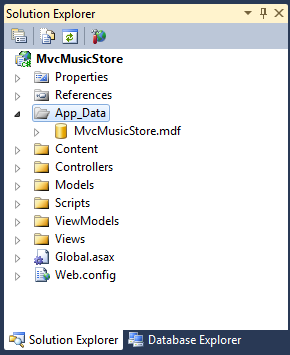
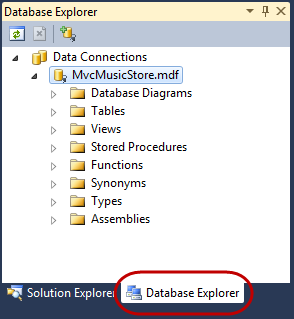
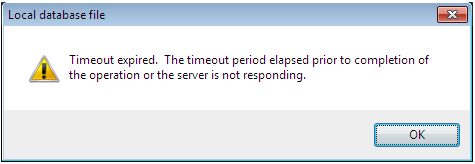
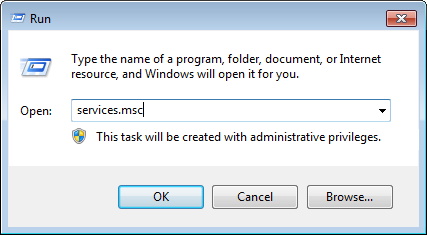
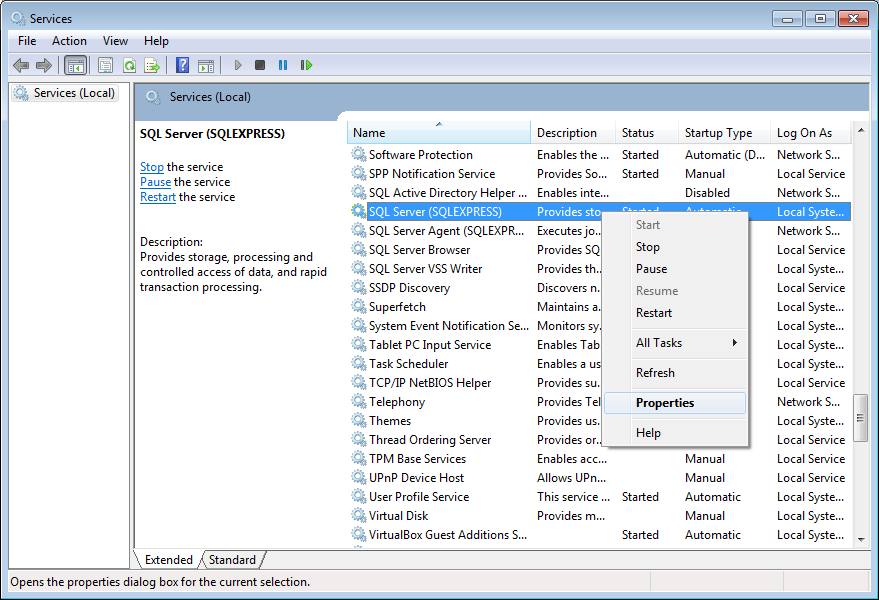
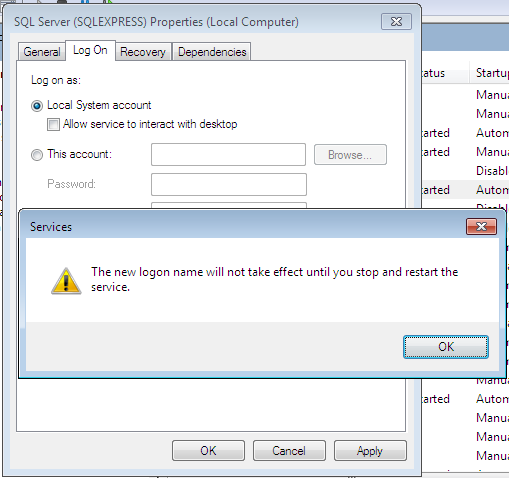
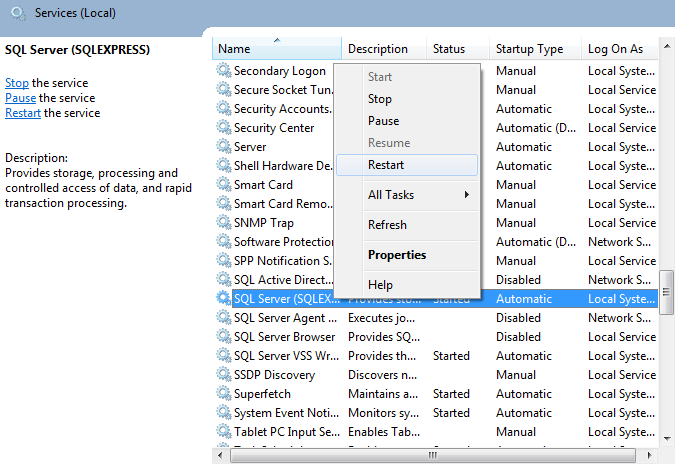
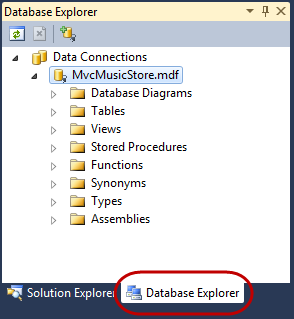
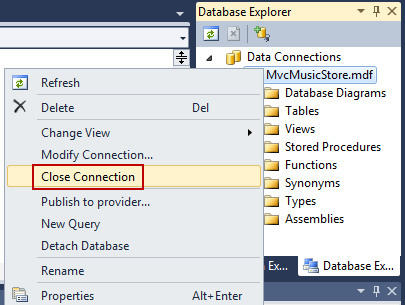
# Next Step

* 1. Exercise 2: Adding a Database Using Code First

Exercise 2: Adding a Database Using Code First

* 1. In this exercise, you will learn how to use the Code First approach to add a database with the tables of the MusicStore application to consume its data.
  2. Once adding the database and generating the model, you will make the proper adjustments in the StoreController to provide the View template with the data taken from the database instead of hardcoding it.
  3. **Note:** If you have completed Exercise 1 and have already worked with Database approach, you will now learn how to get the same results with a different process. Some tasks will be repeated with Exercise 1, so they are marked appropriately to make your reading easier.
  4. If you have not completed Exercise 1 but would like to learn the Code First approach, you can start from this exercise and get a full coverage of the topic.

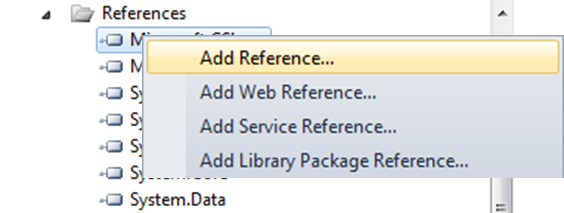
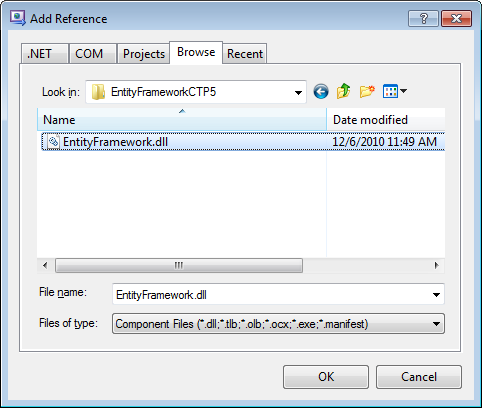
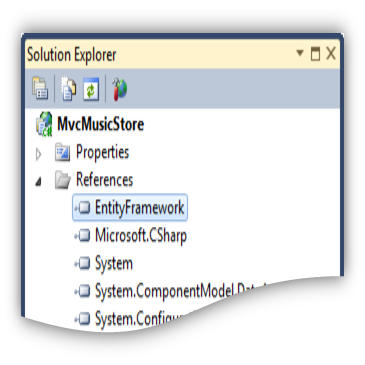
Task 1 – Adding a Database

* 1. In this task, you will add an already created database with the main tables of the MusicStore application to the solution.
  2. **Note:** This task is in common with Exercise 1.
  3. Start Microsoft Visual Web Developer 2010 Express from **Start** | **All Programs** | **Microsoft Visual Studio 2010 Express** | **Microsoft Visual Web Developer 2010 Express**.
  4. In the **File** menu, choose **Open Project**. In the Open Project dialog, browse to Source\Ex02-AddingADatabaseCodeFirst\Begin, select **MvcMusicStore.sln** and click **Open**.
  5. Add an **App\_Data** folder to the project to hold the SQL Server Express database files. **App\_Data** is a special folder in ASP.NET which already has the correct security access permissions for database access. To add the folder, right-click **MvcMusicStore** project, point to **Add** then to **Add ASP.NET Folder** and finally click **App\_Data**.
     1. 
     2. Figure
     3. Adding an App\_Data folder
  6. Add **MvcMusicStore** database file. In this lab, you will use an already created database called **MvcMusicStore.mdf**. To do that, right-click the new **App\_Data** folder, point to **Add** and then click **Existing Item**. Browse to **\Source\Assets\** and select the **MvcMusicStore.mdf** file.
     1. 
     2. Figure
     3. Adding an Existing Item
     4. 
     5. Figure
     6. MvcMusicStore.mdf database file
     7. The database has been added to the project. Even when the database is located inside the solution, you can query and update it as it was hosted in a different database server.
     8. 
     9. Figure
     10. MvcMusicStore database in Solution Explorer
  7. Verify the connection to the database. To do this, open the **Database Explorer** (CTRL+ALT+S), and then double-click the **MvcMusicStore.mdf**. The connection is established.
     1. 
     2. Figure
     3. Connecting to MvcMusicStore.mdf
     4. **Note:** If you getanerror like the following, please follow the steps below.
     5. 
     6. 1. Open the Windows Services console. To do that, open the Run command from **Start** | **All Programs** | **Accessories** | **Run**, type **services.msc** and then click **OK.**
     7. 
     8. **Figure 27**
     9. *Running services.msc*
     10. 2. Right-click the **SQL Server (SQLEXPRESS)** service and select **Properties**.
     11. 
     12. **Figure 28**
     13. *SQL Server (SQLEXPRESS) service*
     14. 3. Open the **Log On** tab, select **Local System account** as the account to log on with and click **OK**. Accept the dialog by clicking **OK** again.
     15. 
     16. **Figure 29**
     17. *Changing the log on account*
     18. 4. Restart the SQL Server (SQLEXPRESS) service.
     19. 
     20. **Figure 30**
     21. *Restarting SQL Server (SQLEXPRESS) service*
     22. 5. Once the service is restarted, close the **Services** console and verify the connection to the database. To do this, select the **Database Explorer**, and then double-click the **MvcMusicStore.mdf**. The connection is established.
     23. 
     24. **Figure 31**
     25. *Connecting to MvcMusicStore.mdf*
  8. Close the connection now. To do that, in Database Explorer right-click on the MvcMusicStore database and select **Close Connection**.
     1. 
     2. Figure
     3. Closing the connection

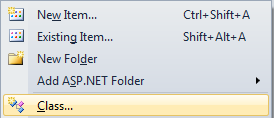
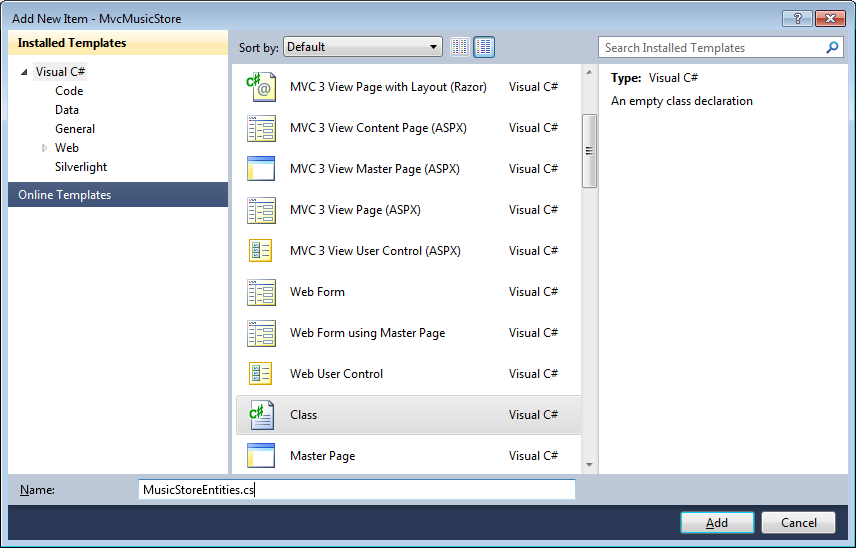
Task 2 – Including Code First in the Solution and Connecting the Database

* 1. Now that we have already added a database to our project, we will write in the Web.config the connection string. Then will include an external library for **Entity Framework 4 Code First.**
  2. **Note:** If you completed Exercise 1, you will note that these steps were automatically generated by a wizard. As Code First is not providing a wizard, you will have to implement many of them manually.
  3. Add a connection string at **Web.config**. To do that, open **Web.config** at project root and include these lines at the end of the file:
     1. 
     2. Figure
     3. Web.config file location
     4. XML
     5. …
     6. **<connectionStrings>**

**<add name="MusicStoreEntities" connectionString="data source=.\SQLEXPRESS;initial catalog=MvcMusicStore;Integrated Security=SSPI;AttachDBFilename=|DataDirectory|\MvcMusicStore.mdf;User instance=true" providerName="System.Data.SqlClient" />**

* + 1. **</connectionStrings>**
    2. </configuration>
  1. Add a new reference to the Code First assembly **EntityFramework.dll.** To do this, right-click **References** and select **Add Reference**:
     1. 
     2. Figure
     3. Adding a reference
  2. In the **Add Reference** dialog, click the **Browse** tab and navigate to **Source\Assets\EntityFrameworkCTP5** and select **EntityFramework.dll**. Then, click **OK**:
     1. 
     2. Figure
     3. Adding EntityFramework assembly
     4. 
     5. Figure
     6. EntityFramework assembly added to the project

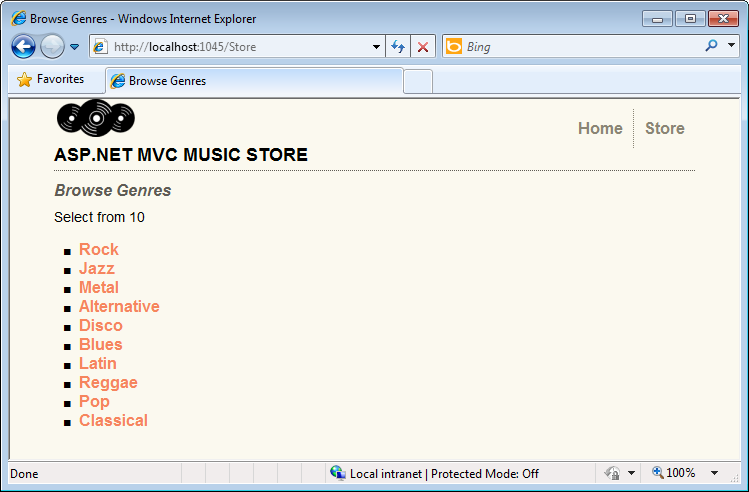
Task 3 – Working with the Model

* 1. Now that we have already included code first library and connected the database, we will link the model with the database tables. In this task, we will create a class that will be linked to the database with Code First. Remember we already have a POCO model class that should be modified.
  2. **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.
  3. Open the POCO model class **Genre** from **/Models** project folder and include an ID, a description attribute, and also an album’s collection.
     1. *(Code Snippet – ASP.NET MVC Models and Data Access – Ex2 Code First Genre – CSharp)*
     2. C#
     3. using System;
     4. using System.Collections.Generic;
     5. using System.Linq;
     6. using System.Web;
     7. namespace MvcMusicStore.Models
     8. {
     9. public class Genre
     10. {
     11. **public int GenreId { get; set; }**
     12. public string Name { get; set; }
     13. **public string Description { get; set; }**
     14. **public virtual ICollection<Album> Albums { get; set; }**
     15. }
     16. }
     17. **Note:** To work with Code First conventions, Genre must have a primary key property that will be automatically detected.
     18. You can read more about Code First Conventions in this [msdn blog article](http://blogs.msdn.com/b/efdesign/archive/2010/06/01/conventions-for-code-first.aspx).
  4. Now, open the POCO model class **Album** from **/Models** project folder and include the AlbumId and GenreId properties.
     1. *(Code Snippet – ASP.NET MVC Models and Data Access – Ex2 Code First Album – CSharp)*
     2. C#
     3. using System;
     4. using System.Collections.Generic;
     5. using System.Linq;
     6. using System.Web;
     7. namespace MvcMusicStore.Models
     8. {
     9. public class Album
     10. {
     11. **public int AlbumId { get; set; }**
     12. public string Title { get; set; }
     13. **public int GenreId { get; set; }**
     14. public virtual Genre Genre { get; set; }
     15. }
     16. }
  5. Right-click the **Models** project folder point to **Add** and then click **Class** toadd a new class**.** Name it **MusicStoreEntities.cs** and click **Add.**
     1. 
     2. Figure
     3. Adding a class
     4. 
     5. Figure
     6. Adding a class(2)
  6. Open the class you have just created, **MusicStoreEntities.cs**, and include the namespace **System.Data.Entity**.
     1. C#
     2. using System;
     3. using System.Collections.Generic;
     4. using System.Linq;
     5. using System.Web;
     6. **using System.Data.Entity;**
     7. **using System.Data.Entity.Database;**
     8. …
  7. Replace the class declaration to extend DbContext class: declare a public DBSet and override the method OnModelCreating. 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:
     1. *(Code Snippet – ASP.NET MVC Models and Data Access – Ex2 Code First MusicStoreEntities – CSharp)*
     2. C#
     3. using System;
     4. using System.Collections.Generic;
     5. using System.Linq;
     6. using System.Web;
     7. using System.Data.Entity;
     8. using System.Data.Entity.Database;
     9. namespace MvcMusicStore.Models
     10. {
     11. **public class MusicStoreEntities : DbContext**
     12. **{**
     13. **public DbSet<Genre> Genres { get; set; }**
     14. **public DbSet<Album> Albums { get; set; }**
     15. **protected override void OnModelCreating(System.Data.Entity.ModelConfiguration.ModelBuilder modelBuilder)**
     16. **{**
     17. **modelBuilder.Conventions.Remove<IncludeMetadataConvention>();**
     18. **modelBuilder.Entity<Genre>().ToTable("Genre");**
     19. **modelBuilder.Entity<Album>().ToTable("Album");**
     20. **base.OnModelCreating(modelBuilder);**
     21. **}**
     22. **}**
     23. }
     24. **Note:** With Entity Framework **DbContext** and **DBSet** you will be able to query the POCO class Genre.
     25. By extending OnModel Creating we are specifying in the **code** how Genre will be mapped to a database table.
     26. You can find more information about DBContext and DBSet in this msdn blog article: [link](http://blogs.msdn.com/b/adonet/archive/2010/09/02/ef-feature-ctp4-dbcontext-and-databases.aspx)

Task 4 – Querying the Database

* 1. In this task, you will update the StoreController class so that instead of using hard-coded data, it can consume it from the database.
  2. **Note**: This task is in common with Exercise 1.
  3. If you completed Exercise 1 you will note these steps are the same, independently from the approach (Database first or Code first). They are different in how the data is linked with the model, but the access to data entities has to be the transparent from the controller.
  4. Open **Controllers\StoreController.cs** and add the following field to hold an instance of the **MusicStoreEntities** class, named **storeDB**:
     1. *(Code Snippet – ASP.NET MVC Models and Data Access – Ex2 Code First storeDB – CSharp)*
     2. C#
     3. public class StoreController : Controller
     4. {
     5. **MusicStoreEntities storeDB = new MusicStoreEntities();**
  5. The **MusicStoreEntities** class exposes a collection property for each table in the database. Update **StoreController**’s **Index** action to retrieve all **Genre** names in the database. This was done previously by hard-coding string data. Now you can instead write a LINQ query expression like the one below which retrieves the **Name** property of each Genre within the database:
     1. *(Code Snippet – ASP.NET MVC Models and Data Access – Ex2 code First Store Index – CSharp)*
     2. C#
     3. //
     4. // GET: /Store/
     5. public ActionResult Index()
     6. {
     7. **// Retrieve the list of genres**
     8. **var genres = from genre in storeDB.Genres**
     9. **select genre.Name;**
     10. // Create your view model
     11. }
     12. **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.
     13. For more information about LINQ, please visit the [msdn site](http://msdn.microsoft.com/en-us/netframework/aa904594.aspx).
  6. Transform the collection of genres to a list. To do this, replace the following code:
     1. *(Code Snippet – ASP.NET MVC Models and Data Access – Ex2 Code First Genres to List – CSharp)*
     2. C#
     3. public ActionResult Index()
     4. {
     5. // Retrieve the list of genres
     6. var genres = from genre in storeDB.Genres
     7. select genre.Name;
     8. // Create your view model
     9. var viewModel = new StoreIndexViewModel
     10. {
     11. **Genres = genres.ToList(),**
     12. **NumberOfGenres = genres.Count()**
     13. };
     14. return View(viewModel);
     15. }

Task 5 – Running the Application

* 1. In this task, you will check that the Store Index page will now display the Genres stored in the database instead of the hard-coded ones. There is no need of changing the View template because the **StoreController** is returning the same **StoreIndexViewModel** as before, although this time the data will come from the database.
  2. Press **F5** to run the Application.
  3. The project starts in the Home page. Change the URL to **/Store** to verify that the list of **Genres** is no longer the hard-coded list, else the ones taken from the database.
     1. ****
     2. Figure
     3. Browsing Genres from the database

# Next Step

* 1. Exercise 3: Querying the Database with Parameters

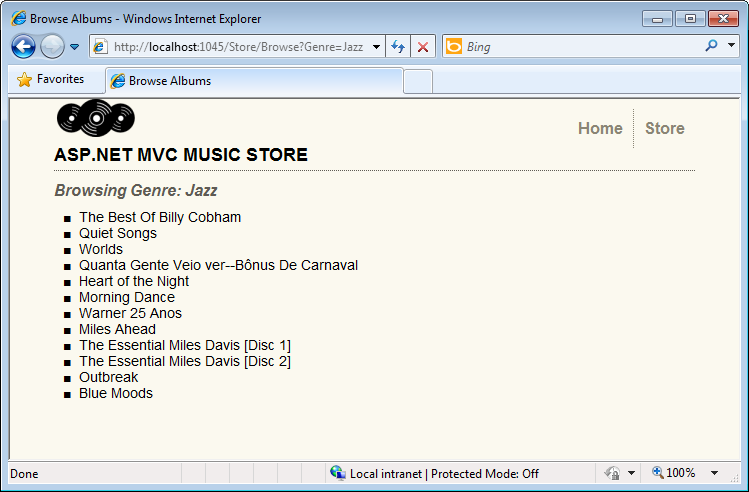
Exercise 3: Querying the Database with Parameters

1. In this exercise, you will learn how to query the database using parameters and how to use the Query Result Shaping, a feature that reduces the number of accesses to the database to retrieve data in a more efficient way.
   1. **Note:** For further information on Query Result Shaping, visit the following [msdn article](http://msdn.microsoft.com/en-us/library/bb896272%28VS.90%29.aspx).

Task 1 – Modifying StoreController to Retrieve Albums from Database

* 1. In this task, you will change the **StoreController** class to access the database to retrieve albums from a specific genre.
  2. If not already open, start Microsoft Visual Web Developer 2010 Express from **Start** | **All Programs** | **Microsoft Visual Studio 2010 Express** | **Microsoft Visual Web Developer 2010 Express**.
  3. In the **File** menu, choose **Open Project**. In the Open Project dialog, browse to Source\Ex03-QueryingTheDatabaseWithParametersDBFirst\Begin (or Ex03-QueryingTheDatabaseWithParametersCodeFirst\Begin if you want to use a Code First approach), select **MvcMusicStore.sln** and click **Open**. Alternatively, you may continue with the solution that you obtained after completing any of the previous exercises.
  4. 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**.
  5. Change the **Browse** action method to retrieve albums for a specific genre. To do this, replace the following code:
     1. *(Code Snippet – ASP.NET MVC Models and Data Access – Ex3 StoreController BrowseMethod – CSharp)*
     2. C#
     3. //
     4. // GET: /Store/Browse?genre=Disco
     5. public ActionResult Browse(string genre)
     6. {
     7. **// Retrieve Genre and its Associated Albums from database**
     8. **var genreModel = storeDB.Genres.Include("Albums")**
     9. **.Single(g => g.Name == genre);**
     10. **var viewModel = new StoreBrowseViewModel()**
     11. **{**
     12. **Genre = genreModel,**
     13. **Albums = genreModel.Albums.ToList()**
     14. **};**
     15. return View(viewModel);
     16. }
     17. **Note:** 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.
     18. **Note:** 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.
     19. 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

* 1. In this task, you will try out the Application in a web browser and obtain albums for a specific genre from the database.
  2. Press **F5** to run the Application.
  3. The project starts in the Home page. Change the URL to **/Store/Browse?genre=Jazz** to verify that the results are being pulled from the database.
     1. 
     2. Figure
     3. Browsing /Store/Browse?genre=Jazz

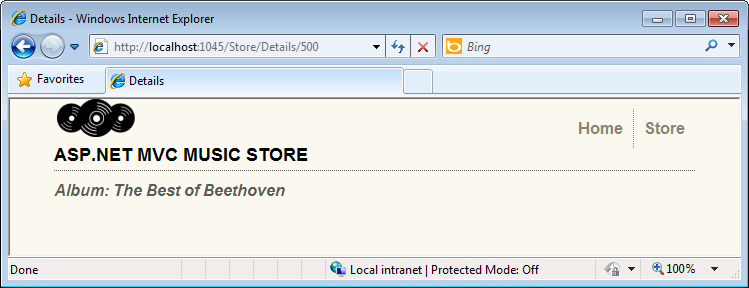
Task 3 – Accessing Albums by Id

* 1. In this task, you will repeat the previous procedure, in this case, to obtain albums by Id.
  2. 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**.
  3. Change the **Details** action method to retrieve albums details based on their **Id**. To do this, replace the following code:

*(Code Snippet – ASP.NET MVC Models and Data Access – Ex3 StoreController DetailsMethod – CSharp)*

* + 1. C#
    2. //
    3. // GET: /Store/Details/5
    4. public ActionResult Details(int id)
    5. {
    6. **var album = storeDB.Albums.Single(a => a.AlbumId == id);**
    7. return View(album);
    8. }

Task 4 – Running the Application

* 1. In this task, you will try out the Application in a web browser and obtain album details based on its Id.
  2. Press **F5** to run the Application.
  3. The project starts in the Home page. Change the URL to **/Store/Details/500** to verify that the results are being pulled from the database.
     1. 
     2. Figure
     3. Browsing /Store/Details/500

# Next Step

* 1. Summary

Summary

* 1. By completing this Hands-On Lab you have learned the fundamentals of ASP.NET MVC Models and Data Access, using a **Database First** approach:
  + How to add a database to the solution in order to consume its data
  + How to update Controllers to provide View templates with the data taken from the database instead of hard-coded one
  + How to query the database using parameters
  + How to use the Query Result Shaping, a feature that reduces the number of accesses to the database to retrieve data in a more efficient way
  + How to use both Database First and Code First approaches in Microsoft Entity to link the database with the model