|  |
| --- |
|  |
| ADO.NET Entity Framework |
| LINQ to Entities |
|  |
|  |
|  |

|  |
| --- |
|  |

# 

LINQ and the Entity Framework

# Overview

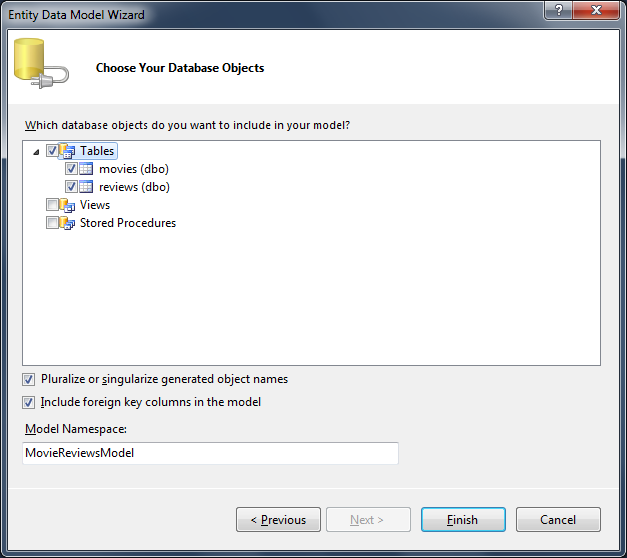
In this lab, we will be exploring the ADO.NET Entity Framework. We will create a new entity data model to interact with a movie review database. We will also see how to modify and update the model after making database schema changes, and use the model to read and update data.

# Part I – Setting Up the Database

1. This lab requires the moviereviews database. If you’ve already installed the database you can move to the next section.
2. To install the database, execute the SQL commands in the Scripts\moviereviews.sql file where you’ve extracted the Pluralsight labs. You can execute the script file using SQL Management Studio, sqlcmd, or any other SQL tool. If you need help, the file Labs\DatabaseSetup\_Troubleshooting.doc file can walk you through the installation.

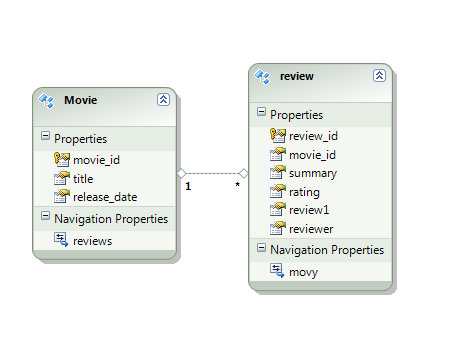
# Part II – Creating an Entity Data Model

1. In Visual Studio, use the File -> Open Web Site command to open the *LINQ\_EntityFramework\before* directory. The before directory is the website we’ll be working on in the lab. The *LINQ\_EntityFramework\after* directory contains a completed version of this lab.
2. Right-click the web site node in the Solution Explorer window and select Add -> New Item.
3. In the Add New Item dialog, select the *ADO.NET Entity Data Model* item. Give the new model the name of *Movies.edmx*. Visual Studio will ask if you want to place the model in the App\_Code directory, answer by clicking *Yes*.
4. In the first screen of the Entity Data Model wizard, highlight the *Generate From Database* icon and click Next.
5. In the next step of the wizard, you’ll need to configure a connection to the *moviereviews* database. The connection may already be available if you’ve previously worked with the database, otherwise, click on the New Connection button to configure the connection. Move to the next step once the connection is ready.
6. In the next step (Choose Your Database Objects), check the checkbox next to the *Tables* entry to select all tables in the database. At the bottom of the dialog, set the *Model Namespace* to *MovieReviewsModel*. Click *Finish*.



1. You should now see the entity designer. Right click on the *Movy* entity and select Properties to open the properties window. Change the ***Name*** property to ***Movie****,* and the ***Entity Set Name*** property to **Movies**.

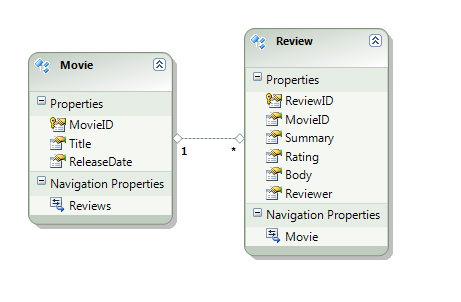
Note: The **Name** property **represents the name of the class** used to represent the entity, and generally is a singular noun. The **Entity Set Name** represents the **name of a property on our entity container that** we use to access the entities, and should generally be the plural form of the entity name. The designer should now look like the following.



1. Continue using the designer to “fix” the names of the entities and their properties. The “movie\_id” property should be “MovieID”, and so on. When naming the “review” property, change the name to “Body” so it does not clash with the entity name of “Review”. Also, **make sure the *Entity Set Name* of the Review entity is set to Reviews**.

Hint: Clicking on a name inside the entity shape and pressing F2 will allow you to use inline editing. You can also click on a name and begin typing to replace the existing name entirely.

1. We also need to change the name of the *Navigation Properties* in each entity. In the Movie entity, we want a property named *Reviews* (capitalized) to find a movie’s associated reviews. In the Review class, we want a property named *Movie* to find a review’s parent (this should not be plural since each review is associated with a single movie). After all the editing is finished, the designer should look like the following.



1. In the Solution Explorer window, right click the Movies.edmx file and select *Open With*. Choose *XML Editor* from the list of editors, and press OK. Answer “Yes” if Visual Studio asks permission to close any files.
2. Inside the edmx file, locate the XML comment that says signifies the start of the *SSDL* content. This comment should be near the top of the file. Find the <*EntityType* > element with a *Name* attribute equal to “movies”.

Notice the language you see to describe the entity is database oriented (nvarchar and datetime Type attributes). Remember, SSDL describes the database that the entity framework will use to persist data for the application.

1. Now locate the XML comment introducing the CSDL content. The XML in this section defines your object model. Locate the *<EntityType>* element for a Movie inside this section. Notice how this section uses CLR datatypes (String, Int32) to describe properties.
2. At the bottom of the XML file is the mapping specification (MSL) that connects the object model (described in the CSDL) to the storage (described in SSDL). You should be able to find the XML that will map the MovieID property to a movie\_id column. It looks like the following:

<EntitySetMapping Name="Movies">

<EntityTypeMapping

TypeName="IsTypeOf(MovieReviewsModel.Movie)">

<MappingFragment StoreEntitySet="movies">

<ScalarProperty Name="MovieID" ColumnName="movie\_id" />

<ScalarProperty Name="Title" ColumnName="title" />

<ScalarProperty Name="ReleaseDate"

ColumnName="release\_date" />

</MappingFragment>

</EntityTypeMapping>

</EntitySetMapping>

Although you might never edit edmx files by hand, it is still a good idea to understand the contents. You might find it useful to write your own tooling for the XML, or perform custom processing that “fixes” names automatically. We will look at some additional editing capabilities in the following steps.

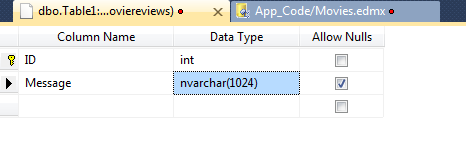
1. Double-click the Movies.edmx file again to open the model in the designer. Answer “Yes” if Visual Studio prompts you to close any files.
2. Find the *Model Browser* window. If the window is not open already, you can open the window from the *View -> Other Windows* menu, or by right clicking in the white space of the designer window and selecting “*Model Browser*”.
3. The Model Browser gives you a different perspective on the CSDL and the SSDL content. You can browse through the entities, associations, tables, and constraints in your model. Locate the EntityContainer node under the MovieReviewsModel.



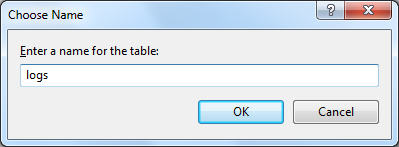
1. Open the EntityContainer node and drill down to expand the *Entity Sets* underneath. If we correctly set the *Entity Set Name* properties in the designer, we should see entity sets named *Movies* and *Reviews*. **If** the names are **incorrect**, use the designer window to change the *Entity Set Name* property of the corresponding entity.
2. Notice our entity container has the name of *moviereviewsEntities*. The entity container is a class we will instantiate to access data, and this name is not appealing. In the model browser, click on the *EntityContainer* to give it focus, and press F4 to open the Properties Window. Give the *EntityContainer* the name of *MovieRepository.*
3. Go to the entity model designer window again and right-click in any white space area of the designer. Notice you can zoom and make grid lines appear from the commands in the context menu. Click “Validate “and ensure that Visual Studio gives you no sign of an error message. Errors will appear in the Errors Window, just as they would appear for a compilation error.

# Part III – Modifying an Entity Data Model

1. Open the Server Explorer window (Ctrl+Alt+S).
2. The server explorer window should already have a connection established to the moviereviews database. If it does not, right-click the Data Connections icon and select *Add Connection*. You’ll need to fill out the standard information to connect to the moviereviews database.
3. Double-click the moviereviews data connection in Server Explorer to view nodes beneath the data connection.
4. Right-click the Tables node and select *Add New Table.* The table designer should appear.
5. Click in the table designer under *the Column Name* header and type *ID*. This is a column name for the first column in the table. Set the *Data Type* to *int*, and uncheck the *Allow Nulls* checkbox.
6. Right-click the new column and select *Set Primary Key*. A small key icon should appear to the right of the new column.
7. Right-click the new column again and select *Properties*. In the Properties Window, set the *Identity Column* property to *ID*.
8. Click in the space underneath the ID column to add a new column with the name of *Message*. Set the *Data Type* for the column to nvarchar(1024). The designer should look similar to the following.



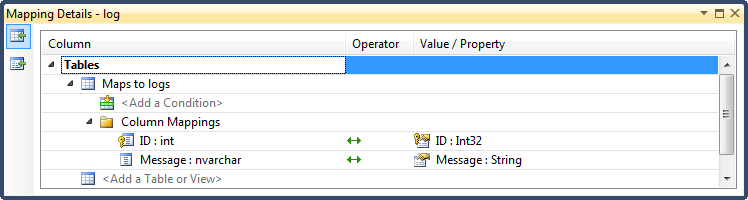
1. Hit Ctrl+S to save the new table. Visual Studio should prompt you for a name – enter *logs*.



1. Return to the Entity Model designer (Movies.edmx).
2. Right-click any white space in the designer and select *Update Model From Database.*
3. In the *Add* tab, select the check box next to the logs table under the Table node. Click Finish.

Notice the designer added a new entity to the design surface with the name of logs.

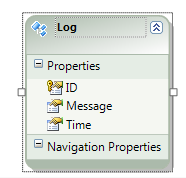
1. Right-click the *Log* entity that we created by hand and select *Table Mapping*. Notice how the designer automatically mapped columns for us.



1. Let’s add another column to the Logs table. Return to the table design view. If you closed the view, go to the Server Explorer window (Ctrl+Alt+S). Double-click the movie reviews connection icon, and expand the Tables node to see all the tables.
2. Right-click the *Logs* table and select *Open Table Definition*.
3. Add a new column to the table named *Time*. Set the *Data Type* to *datetime* and allow null values. Close the designer window and return to the entity designer.
4. Right-click the model and select “*Update Model From Database”.*
5. Go to the *Refresh* tab.

There is actually nothing to do on the Refresh tab since the designer will automatically refresh all mapped tables in the current model. You can only review the list of tables the designer will inspect during the refresh.

1. Click Finish.
2. In the designer model – notice the Log table now has a Time property of type DateTime.
3. Fix up the log table to confirm to the naming standards of C# (capital letters for the type and property names).

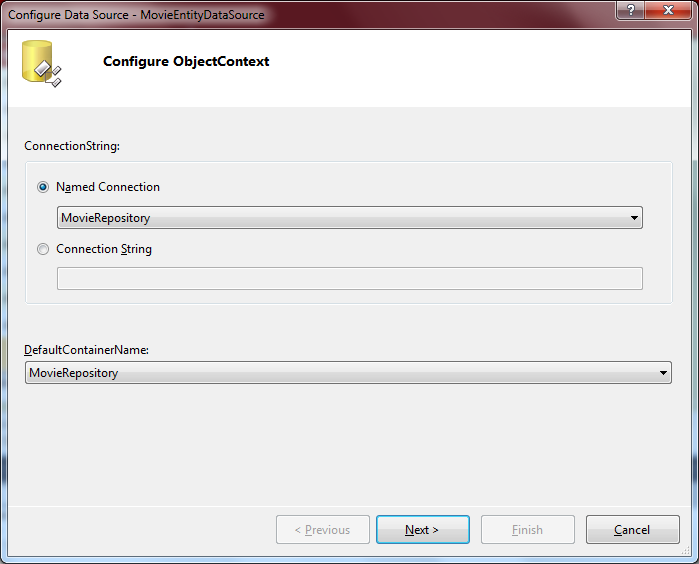


# Using the Model

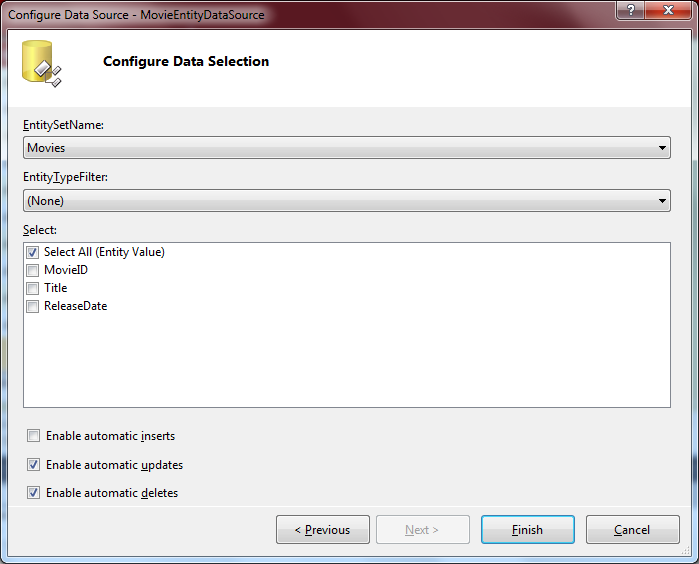
1. Open the website’s Default.aspx file in design view (use the Design / Source / Split buttons on the bottom editor to switch if you do not start with a design view).
2. Notice we have a GridView on the page, but it currently does not have a data source. Open the “GridView Tasks” dialog by clicking on the small chevron that appears in the upper right corner of the GridView when you move the mouse over the control.



1. In the *Choose Data Source* drop down, select *New Data Source*.
2. In the resulting dialog, chose the *Entity* data source type. Give the data source a name of *MovieEntityDataSource*.
3. The next dialog will ask you to configure the Object Context. Select *MovieRepository* as the named connection and DefaultContainerName. Click Next.



1. On the data selection dialog, select *Movies* for the EntitySetName. Click to enable Updates and Deletes on the bottom of the dialog, then click Finish.



1. Back in the GridView Tasks dialog, select the checkboxes to enable Paging, Sorting, Editing, and Deleting.
2. Press F5 to run your project in the debugger. You should be able to use the Grid to browse the movies table, update records, and delete records.



1. Let’s use some C# code to query the database. In the Source view for default.aspx, add a Label control with an ID of *\_topMovie*. Add the label underneath the data source control.

<asp:Label runat="server" ID="\_topMovie"/>

1. Switch to the .cs file for Default.aspx (F7). Add a using statement at the top of the file for the MovieReviewsModel namespace.
2. Inside a check for !PostBack, instantiate a MovieRepository object. Use the Movies and Reviews property of the object to query the database for the movie with the highest average review rating.

The Movies and Reviews properties of the MovieRepository are the “entity sets” we created in the designer. They represent all the Movie and Reviews entities persisted in the database.

1. Set the \_topMovie label’s Text property to the name of the top-ranked movie. The code you’ve written for this should be similar to the following.

if (!IsPostBack)

{

var repository = new MovieRepository();

var topMovie = (

from m in repository.Movies

orderby m.Reviews.Average(r => r.Rating) descending

select m

).First();

\_topMovie.Text = topMovie.Title;

}

1. We can also use repository to update, delete movies in C#. Add the following buttons underneath the label control we added previously.

<asp:Button runat="server" ID="\_updateMovieButton"

Text="Update A Movie" OnClick="UpdateMovie" />

<asp:Button runat="server" ID="\_deleteMovieButton"

Text="Delete A Movie" OnClick="DeleteMovie" />

1. In the .cs file for default.aspx, write an UpdateMovie event handler. The event handler should retrieve the first movie from the MovieRepository. Set the movie’s ReleaseDate to the current date and time, and then use the repository to save changes (using the SaveChanges method).
2. Make sure to call DataBind on the \_movieGrid to update the display.

protected void UpdateMovie(object sender, EventArgs e)

{

var repository = new MovieRepository();

var movie = repository.Movies

.First();

movie.ReleaseDate = DateTime.Now;

repository.SaveChanges();

\_movieGrid.DataBind();

}

1. Write a *DeleteMovie* event handler that will use the *MovieRepository* to delete the second movie in the repository. Hints: you can use the *DeleteObject* method of the repository and the LINQ *Skip* operator.

protected void DeleteMovie(object sender, EventArgs e)

{

var repository = new MovieRepository();

var movie = repository.Movies

.OrderBy(m => m.MovieID)

.Skip(1)

.First();

repository.DeleteObject(movie);

repository.SaveChanges();

\_movieGrid.DataBind();

}

1. Press F5 to run the project with the debugger. You should encounter an exception when you press the Delete button. Click on the *View Detail* link in the exception helper dialog and try to determine the cause of the problem by drilling into the exception.

You should eventually see that the delete operation cannot complete because of a foreign key constraint. The movie still has reviews!

1. To delete all the movie’s reviews, we’ll first load the reviews and setup a loop to mark each one deletion. Add the following code immediately before the call to DeleteObject for the movie.

movie.Reviews.Load();

while (movie.Reviews.Count > 0)

{

repository.DeleteObject(movie.Reviews.First());

}

Note: we can’t use a foreach to call DeleteObject because DeleteObject will remove the review object from the movie’s Reviews collection. It’s illegal to modify a collection while you are iterating the collection. An alternative to manually deleting each review is to setup cascading deletes in the database.

1. Press F5 to run the project. You should now be able to use the Delete button to remove movies from the database.
2. Add the following controls to default.aspx, underneath the buttons we added previously.

<asp:TextBox runat="server" ID="\_logMesssage"

Text="Message" />

<asp:Button runat="server" ID="\_insertLogButton"

Text="Insert A Message" OnClick="InsertMessage" />

1. Back in the .cs file, add an *InsertMessage* event handler. Inside the event handler, instantiate a new Log object and set it’s *Message* property to the contents of the \_logMessage TextBox. Set the object’s *Time* property to the current date and time. Use a *MovieRepository* to insert the new Log. Do **not** set the LogID property.

protected void InsertMessage(object sender, EventArgs e)

{

Log newLog = new Log();

newLog.Message = \_logMesssage.Text;

newLog.Time = DateTime.Now;

var repository = new MovieRepository();

repository.Logs.AddObject(newLog);

repository.SaveChanges();

}

1. Put a breakpoint at the closing bracket of your event handler (F9). Press F5 to run the project in the debugger, and click the button to insert a new Log record.
2. When the debugger stops execution on your breakpoint, inspect the log object you inserted (you can right-click the variable name and select *Quick Watch*).
3. Notice the LogID property of the object is now set. The Entity Framework automatically retrieved the Identity values inserted into the table by the database.
4. You can now stop debugging, or press F5 to continue running and experimenting with your application.

# Conclusion

Congratulations! You’ve finished the lab for the ADO.NET Entity Framework. We’ve seen how to create a new entity data model and modify a model when making database changes. We’ve also used the entity model to create, read, update, and delete data in our movie reviews database. If you want to expand your horizons with the Entity Framework, you can create screens that will allow you to a add, insert, update, and delete reviews, or see if you can create a master detail view in the page we’ve been working with in ASP.NET.