# Data Access

## Estimated time for completion: 45 minutes

## Overview:
In this lab you will learn how to fetch and manipulate using Entity Framework. Rather than write tedious data access code you will use Entity Framework to raise the level of abstraction, replacing low level SQL calls with a more natural object oriented approach.

## Goals:
- Learn how to make data access a breeze with Entity Framework.
- Learn how to perform eager and lazy loading
- Learn how to update entities
- Learn how to invoke stored procedures

## Lab Notes:
*Before commencing the lab you will need to install the pubs database, by running the following command.*

```
osql -E -S <Server Instance> -i instpubs.sql
```

*Verify that the database installed by executing the following*

```
osql -E -S <Server Instance> -dpubs
1> select * from authors
2> go
```

## Part 1: Entity Framework, Simple Queries

*In this part you will be working with Microsoft's ORM (Object Relationship Mapper) Entity Framework. You will use the framework to simplify the creation of data access query code.*

1. Open the project **Before\DataAccess.**sln. Using the pubs database you created above your first task will be to create a list of all publishers. All the publishers are defined in the publishers table.
2. Add a new item to the project of type "ADO.NET Entity Data Model", and call it pubs.edmx.
3. Select the option to Generate Model from database, and select the pubs database, you may have to create a new connection. Expand the Tables section and select the publisher table. Hit Finish and you will be presented with your conceptual model containing a single entity. Capitalise the entity name and properties as per C# naming convention, and rename the pub_id property to Id and pub_name to just Name.

4. The entity set for publishers will be called publishers1 by default this is obviously not a great name. Right click on the design surface and select Model Browser expand the EntityContainer node, and the Entity Sets node and rename the publishers1 entity set to Publishers.
5. Right click on the design surface and select the properties option; the property window will show information about the model. Change the Entity Container Name to **Pubs**. The container name is used as the class name for the generated DbContext, the root object for access to the entity sets. Save it.
6. Expand the pubs.edmx, in the solution explorer, you will see two tt files, these are the template files that produce the entity POCOs. Right click on the **pubs.tt** and run the custom tool, then right click on the **pubs.context.tt** and run the custom tool.
7. Examine the **pubs.Context.cs** and **Publisher.cs** file to see how the conceptual model has been mapped onto the Publisher class and that a Pubs class exists that derives from DbContext.
8. You will have seen that the Publisher class is a partial class allowing further enhancement of the class in a separate file so as not to interfere with code generation. Add a new class file call it PublisherEx.cs and continue the definition of the Publisher class to override ToString(). Make the ToString method return the name of the publisher.

```
public partial class Publisher
{
  public override string ToString()
  {
    return Name;
  }
}
```

9. Now move to Program.cs of the project and write a method that prints out all the publishers. You will need to create an instance of the **Pubs** class this serves as the object context, and wrap it in a using block. Then simply write a foreach loop that iterates through each of the publishers contained in the entity set on the object context called Publishers.
10. Now build a LINQ query that finds all publishers in the USA and use a foreach loop to evaluate the LINQ query and display the publishers.

**Solution: After Part 1\DataAccess.sln**

## Part 2: Entity Framework, Navigation

*In this part of the lab you will extend your conceptual model to include a Title entity, and an association between publisher and titles. You will then write a report to show all the publishers and the titles, loading the titles both eagerly and lazily.*

1. Re-open the conceptual model and right click on the design surface, and select update model from database. Select the titles table and hit finish.
2. Once the conceptual model has been updated modify the title entity to make it conform to C# naming conventions, include in this the navigation properties that have been included in both entities.

3. Moving back to **Program.cs** write a method to display each publisher along with the titles they publish and the price for each title call it **ListPublishersAndTitles**. To do this create a pair of nested foreach loops were by the outer that iterates through each publisher and the inner iterating through each title for a given publisher.
4. Use Sql Profiler to verify that the titles are being loaded in a lazy style.
5. Lazy loading is efficient in terms of memory foot print, but can be inefficient in terms of SQL round trips. Switch off lazy loading inside the context configuration, or via the model browser properties. Re-run the code and verify that the titles are not being loaded. Now introduce eager loading by adding an include clause to the outer foreach to achieve eager loading.

```
pubs.Publishers.Include("Titles")
```

6. Re-run the code and confirm using SQL profiler that the number of queries is reduced.
7. Add a using statement to "System.Data.Entity" and use the version of include that uses a lambda expression.

```
pubs.Publishers.Include(p => p.Titles)
```

**Solution: After Part 2\DataAccess.sln**

## Part 3: Entity Framework, Updates

*In this part you will write some code that will apply a price rise to all titles.*

1. Create a method that will apply a percentage price rise to all titles, with the following signature

```
private static void TitlePriceRise(decimal percentageRise)
```

2. To implement the behaviour using a foreach loop iterate through each title in the Titles entity set and increase its price by the requested percentage. At the end of the iteration make a call to SaveChanges on the object context to persist the changes.
3. Verify these changes are taking place using the **ListPublishersAndTitles** method you created in Part 2.
4. Using Sql profiler, asses if you think this was a good approach for updating all the titles.

**Solution: After Part 3\DataAccess.sln**

## Part 4: Using Stored Procedures

*In this part of the lab you will replace the inefficient update code you wrote in part 3 with a single call to a stored procedure to perform the update.*

1. Rather than fetching all the titles and then updating each one with a specific round trip you will make use of the following stored procedure defined inside the pubs database

```
CREATE PROCEDURE dbo.UpdatePrices
```

```
(  @percentageRise money ) AS

   UPDATE Titles
   set price = price * 1 + @percentageRise / 100

   RETURN
)
```

2. Update the Model from the  Database, ensuring you now include store procedure **UpdatePrices**.
3. Verify that in the Model Browser under Function Imports there is an entry called UpdatePrices.  Rename it to **ApplyPercentagePriceChangeToTitles**, and recompile. There will be a new method on the Pubs class.

```
Pubs.ApplyPercentageChangeToTitles
```

4. To execute this method from your code use the generated methods on the DbContext class called ApplyPercedntageChangeToTitles

**Solution: After Part 4\DataAccess.sln**