# Integrating a Database (w/ EF and LINQ)

.Net Framework with Web Databases
FALL SEMESTER 2019

## Entity Framework

**Entity Framework (EF)** is an **Object Relational Mapping (ORM)** framework produced by Microsoft.

**Object Relational Mapping (ORM)** frameworks are used to make working with databases easier. They hide the actual SQL that is executed and allow the developer to work with the database tables as objects. Without an ORM framework the developer has to code all of the raw SQL Queries into the application and handle all of the type conversions between SQL and C#.

https://www.killerphp.com/articles/what-are-orm-frameworks/

https://en.wikipedia.org/wiki/Object-relational_mapping

## Model Classes

Model classes are generally written as **Plain Old C# Objects (POCO)** as below. Each of which represents a table in the database.

```
public partial class Product
{
    public int ID { get; set; }
    public string Name { get; set; }
    public string Description { get; set; }
    public decimal Price { get; set; }
    public int? CategoryID { get; set; }
}
```

## Data Annotations

You can use the **[Display]** attribute to set a user friendly property name, which will be displayed in all the views that use the model. The **[Display]** attribute is part of the **System.ComponentModel.DataAnnotations** namespace and can be applied to any property in your model. Example:

```
using System.ComponentModel.DataAnnotations

public partial class Category
{
    [Display(Name="Category Name")]
    public string Name { get; set; }
}
```

By default Entity Framework makes a few assumptions about your database:

1. The name of the table is *Things* (where Thing is the name of your model class)
2. The primary key for a table is named Id, ID, or *ThingId* (where Thing is the name of your model class)

Sometimes these assumptions do not hold up, especially when working with existing databases. You can add the **[Table]** attribute to your model class to force the correct table name. You can also add the **[Key]** attribute to specify which column is the primary key. Example:

```
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

[Table("departments")]
public class Department
{
    [Key]
    [Display(Name = "Department #")]
    public string dept_no { get; set; }

    [Display(Name = "Department")]
    public string dept_name { get; set; }
}
```

If one of your tables has a composite key, then you need to apply both the **[Key]** and **[Column]** attributes to every property that is part of the composite key.

```
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

[Table("dept_emp")]
public class DepartmentEmployee
{
    [Key, Column(Order = 0)]
    [Display(Name = "Department #")]
    public string dept_no { get; set; }

    [Key, Column(Order = 1)]
    [Display(Name = "Employee #")]
    public int emp_no { get; set; }

    [Display(Name = "From Date")]
    public DateTime from_date { get; set; }

    [Display(Name = "To Date")]
    public DateTime? to_date { get; set; }
}
```

Entity Framework Code First Data Annotations

https://msdn.microsoft.com/en-us/data/jj591583

Entity Framework Fluent API

https://msdn.microsoft.com/en-us/data/jj591617

Display Attribute

https://msdn.microsoft.com/en-us/library/system.componentmodel.dataannotations.displayattribute(v=vs.110).aspx

## DBContext

To utilize **Entity Framework** you will need to create a subclass of **DbContext** which defines the tables in the database.  For each table in the database, add a property of type **DbSet<T>** where **T** is the model class which represents a row in the table.

```
using System.Data.Entity;
public class StoreContext : DbContext
{
    public DbSet<Product> Products { get; set; }
    public DbSet<Category> Categories { get; set; }
}
```

## Connection Strings

The **Web.config** file is an XML file that can be used to configure a wide variety of settings in your website. (Both during development and after the website is deployed.)

One very important thing to configure in the **Web.config** file are the **connection strings,** which tell the website where to find the database.  The **<connectionStrings>** section contains all of the connection strings for your website. ***Make sure you update the Web.config file in the root of the project and not the one in the Views folder.***

```
<connectionStrings>

    <add name="DefaultConnection"
         connectionString="Data
         Source=(LocalDb)\MSSQLLocalDB;AttachDbFilename=|DataDirectory|\aspnet-BabyStor
         e-20160217123618.mdf;Initial
         Catalog=aspnet-BabyStore-20160217123618;Integrated Security=True"
         providerName="System.Data.SqlClient" />

    <add name="StoreContext"
         connectionString="Data
         Source=(LocalDB)\MSSQLLocalDB;AttachDbFilename=|DataDirectory|\BabyStore.mdf;I
         nitial Catalog=BabyStore;Integrated Security=True"
         providerName="System.Data.SqlClient" />

    <add name="EmployeeDatabase"
         connectionString="Data Source=localhost;Initial Catalog=emp;Integrated
         Security=True"
         providerName="System.Data.SqlClient" />

</connectionStrings>
```

**Data Source** - the database instance that you are connecting to (usually this is the server name)

**AttachDbFilename** - the path the database file (not used when connecting to an actual database server)

**Initial Catalog** - the name of the database

**Integrated Security** - whether or not the web site should connect to the database with the same windows credentials (username + password) as the web site is running under.

**User ID** - the username for database server login (when not using Integrated Security)

**Password** - the password for database server login (when not using Integrated Security)

**Connection Timeout** - the length of time in seconds to wait for a connection to the database server before giving up and throwing an exception.

**MultipleActiveResultSets** - whether or not you can run multiple SQL queries at the same time.

**Pooling** - whether or not the SQL connections should be pooled.

**Max Pool Size** - the maximum number of SQL connections that can be open at one time.

SQL Server Connection Strings for ASP.NET Web Applications

https://msdn.microsoft.com/en-us/library/jj653752(v=vs.110).aspx

SqlConnection.ConnectionString

https://msdn.microsoft.com/en-us/library/system.data.sqlclient.sqlconnection.connectionstring(v=vs.110).aspx

## Connection Pooling

Opening a new connection to a database server can take a few seconds, so it is generally preferable to reuse a small number of connections. .NET manages this automatically by using a pool of open connections. When you need a connection it is taken out of the pool. And when you are done with it is released back to the pool.

## LINQ

"Language-Integrated Query (LINQ) is an innovation introduced in the .NET Framework version 3.5 that bridges the gap between the world of objects and the world of data."

## LINQ Query Syntax Examples

The following sample shows how to retrieve the employees with the last name "Lenart".

```
var query =
    from e in db.employees
    where e.last_name == "Lenart"
    select e;
```

The following sample shows how to retrieve the employees with the last name "Lenart" and the first name "Uri".

```
var query =
   from e in db.employees
   where e.last_name == "Lenart" && e.first_name == "Uri"
   select e;
```

The following sample shows how to retrieve all the "Senior Staff" with a last name of "Lenart".

```
var query =
    from e in db.employees
    join t in db.titles on e.emp_no equals t.emp_no
    where e.last_name == "Lenart" && t.title == "Senior Staff"
    select new
    {
        emp_no = e.emp_no,
        first_name = e.first_name,
        last_name = e.last_name,
        title = t.title,
        title_from_date = t.from_date,
        title_to_date = t.to_date,
    };
```

The following sample shows how to retrieve all employees in the "Sales" department with a last name of "Lenart".

```
    var query =
        from d in db.departments
        join de in db.dept_emp on d.dept_no equals de.dept_no
        join e in db.employees on de.emp_no equals e.emp_no
        where d.dept_name == "Sales" && e.last_name == "Lenart"
        select new
        {
            first_name = e.first_name,
            last_name = e.last_name,
        };
```

The following sample shows how to retrieve all the titles that have been held by any employee. Distinct() is used to remove duplicate entries from the results.

```
    var query =
        (from t in db.titles select t.title).Distinct();
```

The following sample shows how to retrieve the employees with the last name "Lenart" and order the result by first name in ascending order.

```
 var query =
     from e in db.employees
     where e.last_name == "Lenart"
     orderby e.first_name
     select new { e.emp_no, e.first_name };
```

The following sample shows how to retrieve the employees with the last name "Lenart" and order the result by first name in descending order.

```
 var query =
     from e in db.employees
     where e.last_name == "Lenart"
     orderby e.first_name descending
     select new { e.emp_no, e.first_name };
```

The following sample shows how to retrieve the average salary for employees with the last name "Lenart". This query uses a group by clause to group the salaries that employees have held over time by employee number.

```
var query =
    from e in db.employees
    join s in db.salaries on e.emp_no equals s.emp_no
    where e.last_name == "Lenart"
    group new { e.emp_no, e.first_name, s.salary } by e.emp_no into g
    select new
    {
        emp_no = g.Key,
        first_name = g.FirstOrDefault().first_name,
        avg_salary = Math.Round(g.Average(x => x.salary))
    };
```

## LINQ Methods (requires using System.LINQ)

Filtering

| Function | Description |
|---|---|
| Where(*x* => *clause*) | Returns the sequence of items where the clause is true. |
| First() | Returns only the first item. Throws an exception if there are no items. |
| FirstOrDefault() | Returns only the first item. Returns null if there are no items. |
| First(*x* => *clause*) | Returns only the first item where the clause is true. Throws an exception if there are no items where the clause is true. |
| FirstOrDefault(*x* => *clause*) | Returns only the first item where the clause is true. Returns null if there are no items where the clause is true. |

Counting

| Function | Description |
|---|---|
| Any() | Returns true if there any items in the result set. |
| Any(*x* => *clause*) | Returns true if there any items in the result set where the clause is true. |
| All(*x* => *clause*) | Returns true if the clause is true for all items in the result set. |
| Count() | Returns the number of items in the result set. |
| Count(*x* => *clause*) | Returns the number of items in the result set where the clause is true. |

Aggregating

| Function | Description |
|---|---|
| Sum(*x* => *column*) | Returns the sum for a particular column. |
| Average(*x* => *column*) | Returns the average value for a particular column. |
| Min(*x* => *column*) | Returns the minimum value for a particular column. |
| Max(*x* => *column*) | Returns the maximum value for a particular column. |

Sorting

| **Function** | **Description** |
|---|---|
| OrderBy(*x* => *column*) | Sorts the result set by a particular column in ascending order. |
| OrderByDescending(*x* => *column*) | Sorts the result set by a particular column in descending order. |
| ThenBy(*x* => *column*) | Used to sort the results by multiple columns. |
| ThenByDescending(*x* => *column*) | Used to sort the results by multiple columns. |

Other Methods

| **Function** | **Description** |
|---|---|
| Select(*x* => *result*) | Projects each item into a new form. Often used when you only want some of the columns in the result set. |
| Skip(*n*) | Discards the first *n* items from the result set and keeps the rest. |
| Take(*n*) | Gets the first *n* items from the result set and discards the rest. |
| Distinct() | Removes duplicates from the result set.<br><br>WARNING: Distinct() is expensive in terms of execution time and memory, when there are few duplicates. If you can modify your query to remove duplicates some other way, that is often preferable. Duplicate results are often a sign that your joins are incorrect. |
| ToList() | Execute the query and store it in a list.<br><br>WARNING: If you call ToList() too early, then you will be downloading a lot more data than you need, which will drastically hurt the performance of your application. |

## Async Methods (requires using System.Data.Entity)

*All the non-deferred methods have async versions.*

*Note that all of these methods require an IQueryable<T>*

- SaveChangesAsync()
- ToListAsync()
- FirstAsync(), FirstOrDefaultAsync(), etc
- AnyAsync(), AllAsync(), CountAsync(), etc
- SumAsync(), AverageAsync(), MinAsync(), MaxAsync()

https://docs.microsoft.com/en-us/dotnet/api/system.data.entity.queryableextensions?view=entity-framework-6.2.0

## Further Reading on LINQ

Introduction to LINQ
https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/linq/introduction-to-linq

Introduction to LINQ Queries
https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/linq/introduction-to-linq-queries

C# Features That Support LINQ
https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/linq/features-that-support-linq

LINQ Query Syntax Examples
https://msdn.microsoft.com/en-us/library/gg509017.aspx

Method-Based LINQ Query Examples
https://docs.microsoft.com/en-us/dotnet/framework/data/adonet/method-based-query-examples-linq-to-dataset