Sheridan | Get Creative

Advanced .NET Server Development: Entity Framework (EF)

Instructor: Amandeep S. Patti

Learning Outcome

- Implement data access code using Microsoft Entity Framework and LINQ to Entities

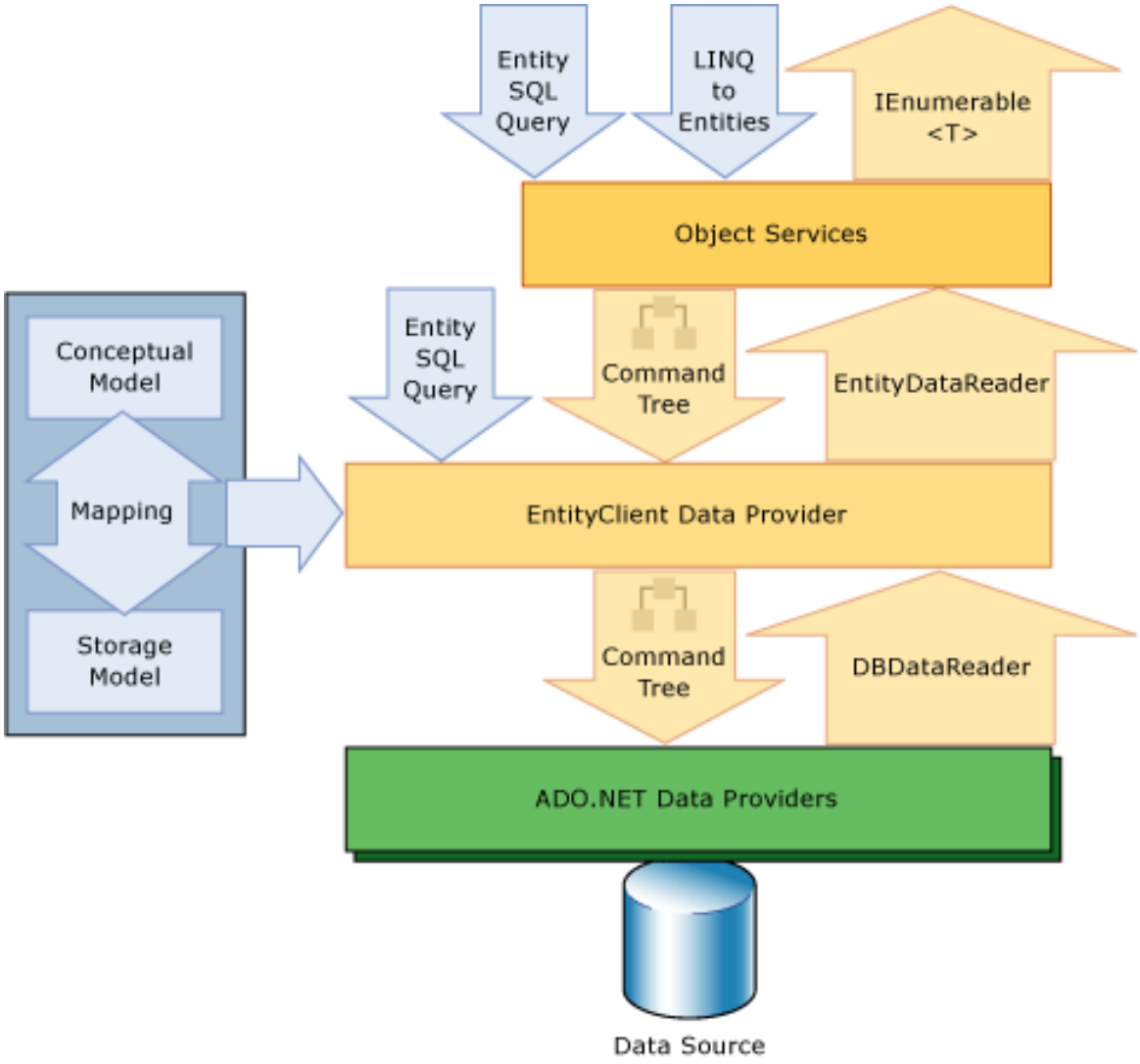
Objectives

- Introduction to Entity Framework
- Entity Data Model
- Entity Framework and Web Forms
- LINQ to Entities
- Entity Framework Workflows

Introduction to Entity Framework

- Open Source ORM framework for ADO.NET
- Part of .NET Framework
- Allow us to choose a workflow based on our requirements and preference from:
  - Database First / Model First / Code First
- Changes to entities are tracked automatically by the context
- Easily save all changes to our entities using the SaveChanges method of our DbContext derived context
- We can query our entities using LINQ to Entities

Entity Framework Architecture



"Entity Framework Overview." *MSDN – the Microsoft Developer Network*. N.p., n.d. Web. 24 Aug. 2013. <http://msdn.microsoft.com/en-us/library/bb399567.aspx>.

Entity Data Model

- Entity Data Model (EDM) describes:
  - Conceptual Layer
    - The model of the conceptual data entities / .NET entity classes
  - Logical Layer
    - The underlying physical data store / relational data
  - Mapping Layer
    - The mapping between the conceptual and physical models
- It does this using 3 XML markup languages:
  - CSDL (Conceptual Schema Definition Language)
  - SSDL (Store Schema Definition Language)
  - MSL (Mapping Specification Language).

Visual Studio EDM Designer

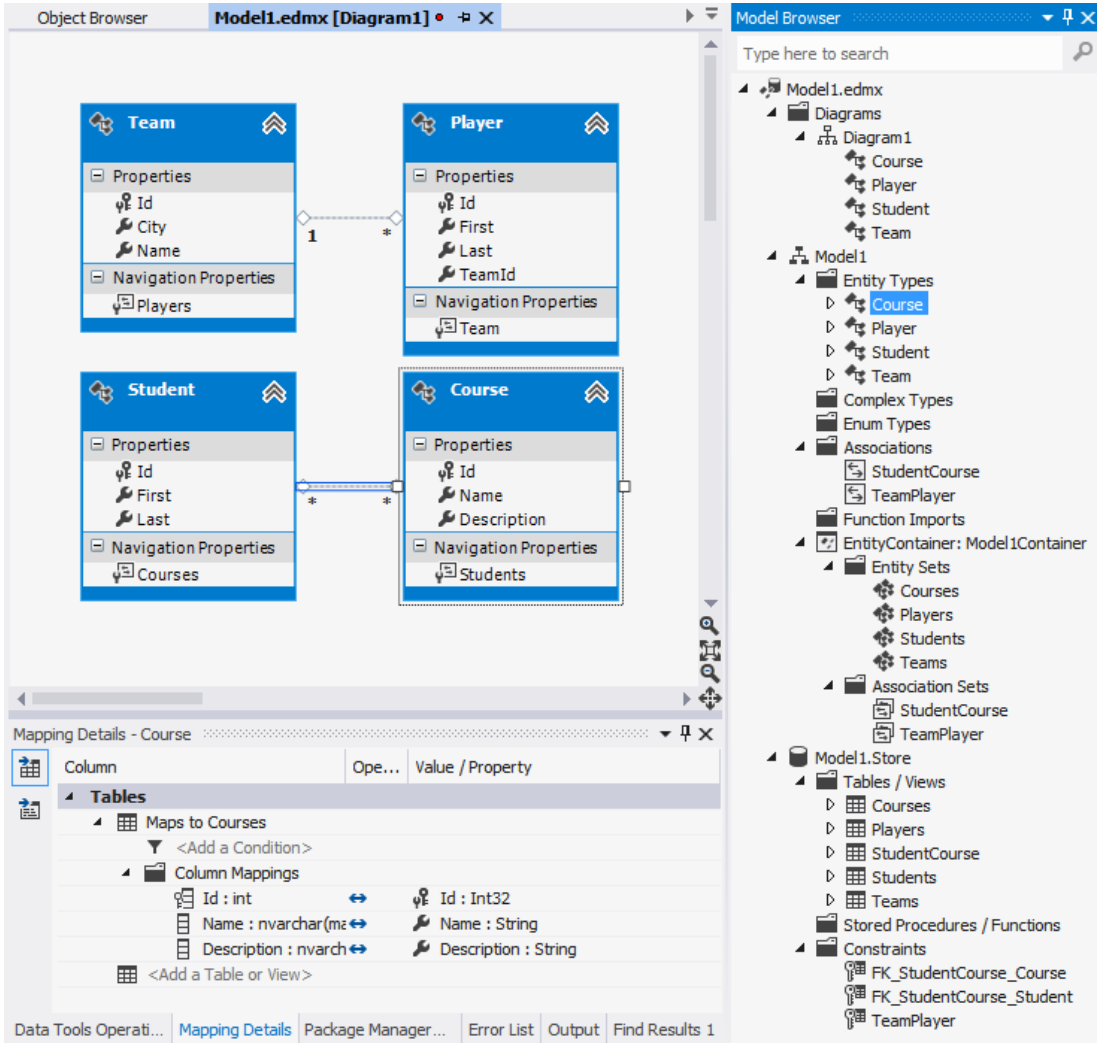
- Visually create EDM and mapping specification
- The output of the tool is XML file (*.edmx)
  - Specifying the schema and the mapping
  - Contains EF metadata artifacts (CSDL/MSL/SSDL content)
  - CSDL/MSL/SSDL) can also be created or edited manually
- Connection String
  - Stored in the configuration file
  - Different from the normal ADO.NET
    - As mapping information is required

Visual Studio EDM Designer

- Graphical representation of the EDM model
  - For the conceptual entities
- It uses associations to model relations
- It provides three types of properties
  - Scalar for primitive types
  - Navigation for relations
  - Complex for grouping properties
- Can easily generate a database from the Data Model
  - by right clicking on the Designer and selecting "Generate Database from Model..."
- Classes are automatically generated from the Data Model
  - on building the project.

Visual Studio Designer

- Note the 3 main areas:
  - Designer
  - Model Browser
  - Mapping Details
- Note the differences between relationships:
  - One to Many
  - Many to Many
- Note the 3 main nodes of the Model Browser:
  - Diagrams
  - Conceptual Model
  - Data Store



Key Classes

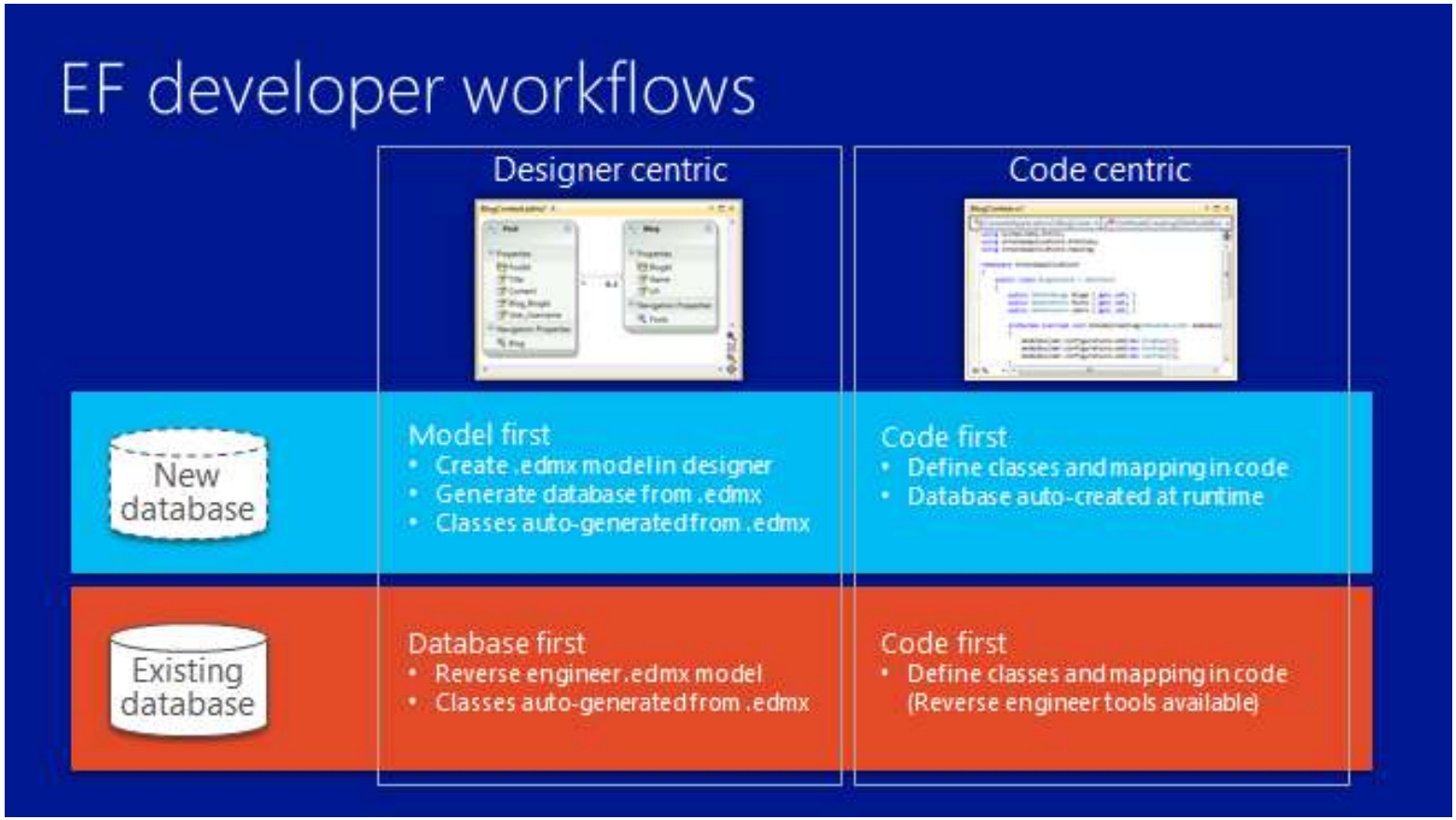
- Main "Context" container class provides access to all of our entities
  - Derives from DbContext
  - Contains DbSet<T> members for our strongly typed entities
  - Provides support for change tracking to our entities
  - Part of the DbContext API which simplified the older ObjectContext API
- Our entities with the DbContext API are POCO (Plain Old CLR Objects) and therefore are persistence ignorant.
- The classes that the Entity Model Designer creates are partial and designed to be easily extended.

Entity Framework Workflows

- Database First
  - Reverse engineer an existing database
  - Commonly used in large organizations with existing databases
- Model First
  - Ideal when starting from scratch
  - Easy to share with multidisciplinary teams
  - Enforces design discipline
- Code First
  - Can be used for existing databases or new databases
  - Ideal for small projects, experienced developers and/or those who believe only code matters
  - Support for updating database schemas without losing data

Entity Framework Workflows

- http://msdn.microsoft.com/en-us/data/jj590134.aspx



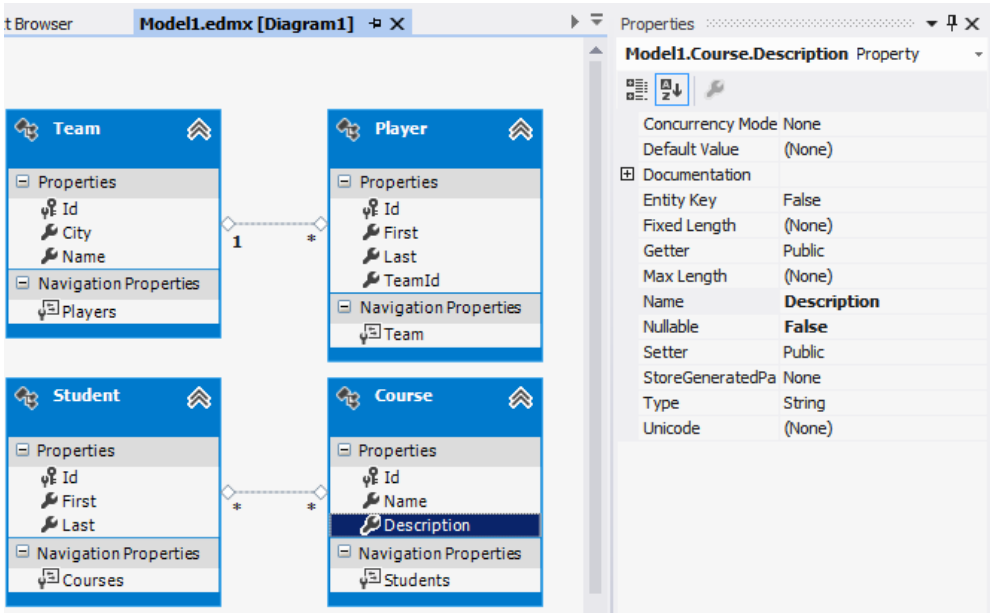
Miller, Rowan. "Building Data Centric Applications for Web, Desktop and Mobile with Entity Framework 5." *Channel 9*. N.p., 30 Oct. 2012. Web. 05 Oct. 2013.

Model First Workflow

- Designed for new projects without an existing database
- Develop a conceptual model using a graphical designer
- Designer can generate database and code from this conceptual model
- Designer can update the model at any time
  - The database will be deleted and regenerated each time the updates are published to the database
- Helps with the design process
- Ideal for the Design Phase of your Capstone project

Model First Workflow

- When a property is selected, the Properties Window allows us to set its properties (length, type, etc.)
- Properties can be grouped into complex types in the conceptual model for convenience
  - But then they are only visible from the Model Browser.



Model First Workflow

- http://msdn.microsoft.com/en-us/data/jj205424.aspx

Entity Framework – Walkthroughs \ Videos

- Entity Framework Database First
  - https://msdn.microsoft.com/en-us/library/jj206878(v=vs.113).aspx
- Entity Framework Model First
  - https://msdn.microsoft.com/en-us/library/jj205424(v=vs.113).aspx
- Exploring the Classes Generated from an Entity Data Model
  - https://www.youtube.com/watch?v=9GGNWywNUhM
- Consuming an Entity Data Model from a separate .NET Project
  - https://www.youtube.com/watch?v=mhs3k4WHwVU

LINQ to Entities

- By default EF uses Lazy Loading which means
  - queries are often not executed during the LINQ statement but instead when an Execution Trigger occurs such as iterating through the results.
- We can use LINQ to query our entities just as you have used LINQ to query objects and SQL Server in the past.
- LINQ is powerful and easy to use.

```
13          using (var db = new Model1Container())
14          {
15              var courses = from course in db.Courses
16                            select course;
17
18              // lazy loading means query doesn't execute until we iterate here!
19              foreach (Course course in courses)
20                  Console.WriteLine("ID: {0}, Name: {1}", course.Id, course.Name);
21          }
```

LINQ to Entities

- Instead of Query Syntax you can also use Method Syntax which is even more powerful:

```
23          using (var db = new Model1Container())
24          {
25              var advDotNet = from course in db.Courses
26                              where course.Name.ToUpper().Contains(".NET") && course.Name.Contains("Advanced")
27                              orderby course.Name
28                              select new { course.Name, course.Id };
29
30              // lazy loading means query doesn't execute until we iterate here!
31              foreach (var course in advDotNet)
32                  Console.WriteLine("ID: {0}, Name: {1}", course.Id, course.Name);
33          }
```

- Here is a more complex LINQ statement that adds filtering, sorting and a projection (creating a new anonymous type based on the range variable's fields)

```
35          using (var db = new Model1Container())
36          {
37              var advDotNet = db.Courses.Where(c => c.Name.ToUpper().Contains(".NET") && c.Name.Contains("Advanced"))
38                                        .OrderBy(c => c.Name)
39                                        .Select(c => new {c.Name, c.Id});
40
41              // lazy loading means query doesn't execute until we iterate here!
42              foreach (var course in advDotNet)
43                  Console.WriteLine("ID: {0}, Name: {1}", course.Id, course.Name);
44          }
```

LINQ to Entities

- To add/create a new record to the database:

```
46              using (var db = new Model1Container())
47              {
48                  Course newDotNet = new Course() { Name = "Advanced .NET Server Development",
49                                                    Description = "The most awesome course ever at Sheridan!" };
50                  db.Courses.Add(newDotNet);
51                  db.SaveChanges();
52              }
```

- To update an existing record in the database:

```
66              using (var db = new Model1Container())
67              {
68                  var needsUpdate = from course in db.Courses
69                                    where course.Id == 6
70                                    select course;
71                  foreach (Course course in needsUpdate)
72                    course.Description = "Hurray, I just changed the description of this course!";
73                  db.SaveChanges();
74              }
```

- To remove an existing record from the database:

```
54              using (var db = new Model1Container())
55              {
56                  var cancelled = from course in db.Courses
57                                  where course.Id == 5
58                                  select course;
59                  if (cancelled.Count() > 0)
60                  {
61                      db.Courses.Remove(cancelled.First());
62                      db.SaveChanges();
63                  }
64              }
```

LINQ to Entities

- Changes to entities are tracking automatically. We just need to remember to call SaveChanges on the context ("db" above) when we're finished!

LINQ to Entities

- Getting Started with LINQ to Entities
  - http://msdn.microsoft.com/en-us/data/ff628210.aspx

EF Exception Handling

- EF does throw exceptions in addition to exceptions thrown by the .NET framework and the Data Provider (ex. SQL Server)
- Make sure that your code is exception safe!
- All statements where the DB Context is in scope should be placed in a try catch "catch all" block
- You may also want to handle specific exceptions separately (ex. OptimisticConcurrencyException)
- Remember to have an exception handling strategy including notifying the user of exceptions and logging them with the following information:
  - what operation failed
  - why it failed (the exception)
  - when it failed (log only)
  - who it failed for (log only)

EF Exception Handling

```
try
{
    using (var db = new Model1Container())
    {
        var courses = from course in db.Courses
                      select course;

        // lazy loading means query doesn't execute until we iterate here!
        foreach (Course course in courses)
            Console.WriteLine("ID: {0}, Name: {1}", course.Id, course.Name);
    }
}
catch (Exception ex)
{
    Console.WriteLine("The request for available courses failed due to the following exception: "
                      + ex.Message);
    string exceptionMsg = string.Format("At {0}, {1}'s request for available courses failed due "
                                        + "to the following exception: {2}",
                                        DateTime.Now, userName, ex.Message);
    Trace.WriteLine(exceptionMsg, "Exceptions");
}
```

Web Forms and Entity Framework

- Entity Framework can be used with ASP.NET Web Forms
- WebForms provides the EntityDataSource control to work with the visual Data Controls (GridView, DetailsView, etc.)
- IDE provides less default code (Scaffolding) support than in ASP.NET MVC.
- LINQ to Entities can be used to query our context to fetch, order, group, etc. a set of entities

Database First Workflows

- Designed to reverse engineer an existing database
- Automatically generates an EDMX model and code from an existing database
- EDM Designer allows us to update the model from database at any time as the database changes
- Can upload tables, views or stored procedures from database

Database First Workflows - Video

- Entity Framework Database First
  - http://msdn.microsoft.com/en-us/data/jj591506

Code First Workflow

- Code centric approach to database design which avoids the Designer completely
- Very cool feature. Has generated the most "buzz" lately.
- Provides Code First Data Migrations which allows you to update a database schema without losing any data! Database initializers (Database.SetInitializer()) can also be used to update schema but this causes data loss
- Doesn't show the logical model unless
  - You generate it with your program
  - You use the Entity Framework Power Tools Add-on
- Remember to set your navigation properties as public and virtual to enable Lazy Loading!

Code First Workflow & Capstone

- Code First Workflow can be dangerous as it encourages developers to skip the Design phase
- Avoid this approach for the Design Phase (semester 5) of your Capstone project!
- Consider this approach for the Implementation Phase (semester 6) of your Capstone project as you can do Code First from an existing database!
- You can update the schema of an existing database without losing any data with Data Migrations!

Code First

- Entity Framework Code First to a New Database
  - http://msdn.microsoft.com/en-us/data/jj572366.aspx

EF Data Validation

- Common ways to add support for EF Data Validation
  - Code First Data Annotations (ex. [Required])
  - Adding an associated metadata class or "buddy class" with annotations (for Model First or Database First)
  - Implementing the IValidatableObject interface for the model class (aka self-validating models) for more complicated validation (ex. illegal combinations of values)

EF Data Validation – Annotations

- The most common approach
- This approach is DRY (Don't Repeat Yourself) and works great with ASP.NET MVC!
- In MVC, we can annotate our model classes and have the corresponding validation automatically used in the View (Html Helpers); Database and the Controller update code (ie. ModelState.IsValid)
- Common Data Validation Attributes include:
  - Required
  - StringLength
  - Range
  - DataType
  - RegularExpression
- You can also define custom validation attributes by deriving from the ValidationAttribute class

More on EF Annotations

- Data Annotations can also be used to override the default configuration and mapping options for Code First (ex. [Key])
- The Fluent API is a more advanced approach that allows us to do this in code.

EF Data Validation – Buddy Class

- Used in Model First or Database First Workflows
- Adding a buddy class with annotations

```
[MetadataType(typeof(CourseMetaData))]
public partial class Course {
    public class CourseMetaData {
        [StringLength(50), Required]
        public object Name { get; set; }
    }
}
```

- This approach is NOT completely DRY since we will end up repeating properties in the buddy class to add annotations

EF Data Validation

- Implementing IValidatableObject interface
  - Implement the IValidatableObject interface (aka self-validating models)
    - public class Course: IValidatableObject {
  - Implement the Validate method which is automatically invoked for each modified entity if it is implemented when the DbContext.SaveChanges method is called
    - public IEnumerable<ValidationResult> Validate(ValidationContext validationContext)
    - {
    - if (StartDate.Date >= EndDate.Date)
    - {
    - yield return new ValidationResult("Start Date must be earlier than End Date",
    - new[] { "StartDate", "EndDate" });
    - }
    - }
    - Lerman, Julia, and Rowan Miller. Programming Entity Framework. Sebastopol, CA: O'Reilly Media, 2012. Print.

Code First Data Migrations

- To update the schema of an existing database you need to use Data Migrations.
- Data Migrations allow you to change the schema of your database without any data loss!
- From the Package Manager Console ensure that the "Default project" is set correctly and then enter the following commands:
  - Enable-Migrations (once your baseline database is established)
  - Add-Migration [MyMigrationName] (when you want to add support for a schema change)
  - Update-Database (when you want to perform the database migration)

EF Inheritance Models

- By default Entity Framework using a Table Per Type (TPT) hierarchy for modeling inheritance
- Table Per Hierarchy is also supported and can provide some performance improvements but leads to poorly designed and normalized databases
- Use the default Table Per Type (TPT) for your Capstones!

T4

- Stands for Text Template Transformation Toolkit
- Code generation tool used by Visual Studio
- File extension is ".tt"
- The EDM Designer uses T4 to generate the model code from the conceptual model
- Allow developers to control the code the designer generates

EF Tips

- Put your data model and associated code in a separate Class Library project or at a minimum a separate folder.
- Remember to handle exceptions when working with EF
- Avoid complex types in your model when using Model First as they are not displayed in the designer. Use separate entities instead.
- Use Model First for new projects as this enforces best practices and provides a visual conceptual model to share with your team.
- Be aware of Lazy Loading when using LINQ to entities
- Embrace LINQ to Entities and use LINQPad to experiment with LINQ
- Use the SQL Server Profiler to examine the SQL queries that LINQ to Entities is generating
- Consider using Stored Procedures in your database since they are supported in EF and provide the usual benefits
- Use Database First if you plan on using lots of Stored Procedures although Code First does provide some support for this as well.

Videos

- Code-first development with the Entity Framework
  - http://www.lynda.com/ASPNET-tutorials/Code-first-development-Entity-Framework/158377/171749-4.html
- Database-first development with the Entity Framework
  - http://www.lynda.com/ASPNET-tutorials/Database-first-development-Entity-Framework/158377/171750-4.html
- Using LINQ to Entities
  - http://www.lynda.com/ASPNET-tutorials/Using-LINQ-Entities/158377/171751-4.html