Microsoft® Official Course

Module 7

Accessing a Database

Microsoft®

Module Overview

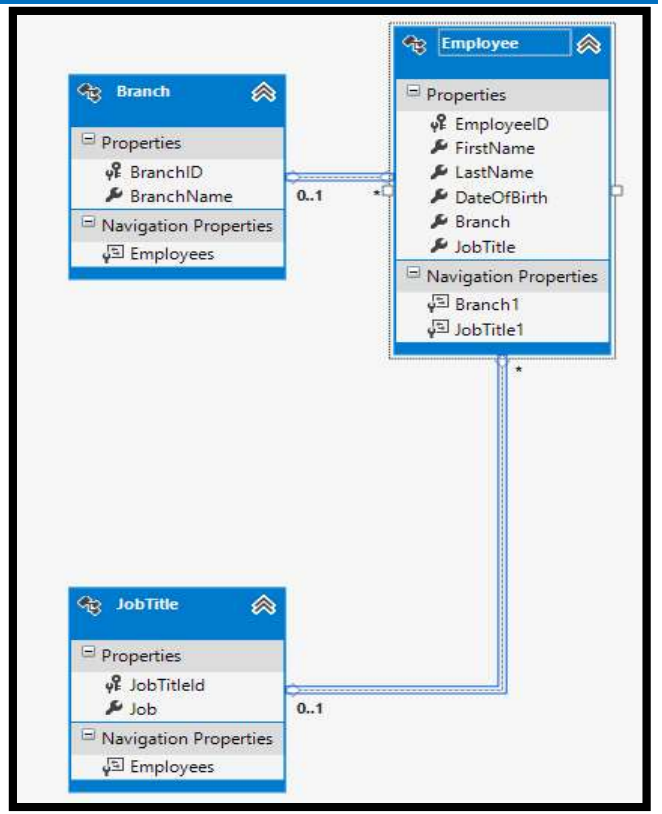
- Creating and Using Entity Data Models (EDM)
- Querying Data by Using Language-Integrated Query (LINQ)

Lesson 1: Creating and Using Entity Data Models

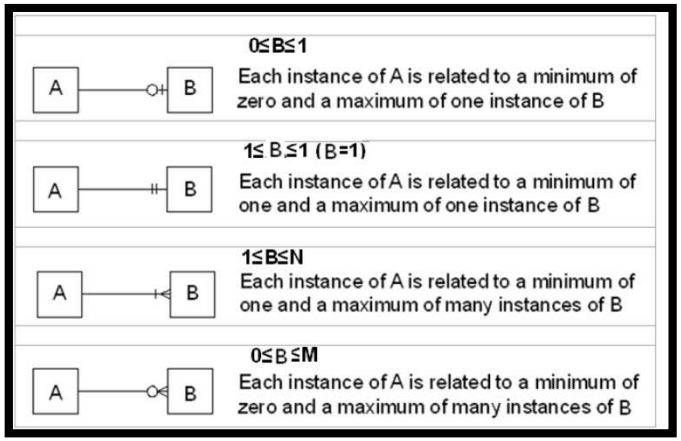
- Introduction to the ADO.NET Entity Framework
- Using the ADO.NET Entity Data Model Tools
- Demonstration: Creating an Entity Data Model
- Customizing Generated Classes
- Reading and Modifying Data by Using the Entity Framework
- Demonstration: Reading and Modifying Data in an EDM

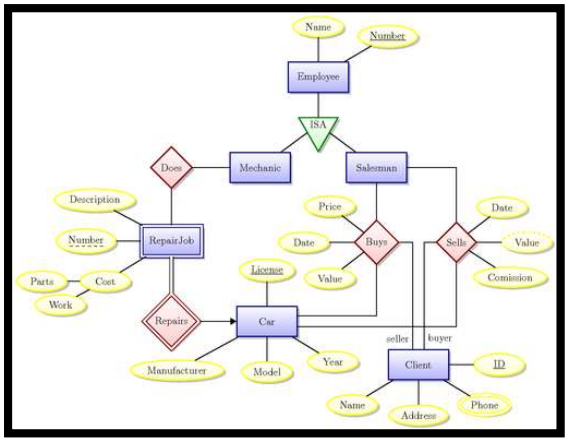
Entity-Relationship Diagram (ERD) – just an intro
source: Google + CSC 446 …

- based on the notion of real world entities and relationship among them
- creates a set of entities with their attributes, a set of constraints and relation among them.
- is best used for the conceptual design of database



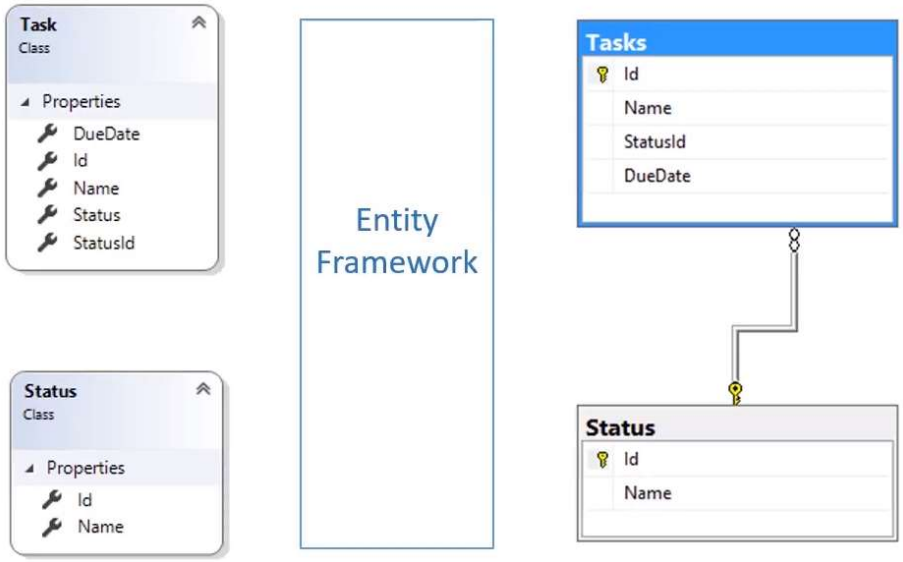
- can be represented as follows:
  - **Entity** - An entity is a real world being, which has some properties called **attributes**.
    - Every attribute is defined by its corresponding set of values, called **domain**.
      - For example, Consider a school database. Here, a student is an entity. Student has various attributes like name, id, age and class etc.
  - **Relationship** - The logical association among entities.
    - Relationships are mapped with entities in various ways.
    - Mapping cardinalities define the number of associations between two entities.
    - Mapping cardinalities:
      - one to one
      - one to many
      - many to one
      - many to many





Extra

- You may want to watch these videos:
  - https://www.youtube.com/watch?v=S9HrLdSrVho
    "Entity Framework - Part 0 - Introduction"
    - C# classes (the model classes)
    - SQL Tables (database objects)
  - https://dotnetplaybook.com/how-much-did-the-mcsd-cost-me/



Introduction to the ADO.NET Entity Framework

This topic is designed to give a brief overview of the ADO.NET Entity Framework, so do not go into a great level of detail. The salient points are expanded upon later in the module.

- The ADO.NET Entity Framework provides:
  - EDMs (Entity Data Models):
    - models that you can use to map database tables and queries to .NET Framework objects
  - Entity SQL:
    - storage independent query language that enables you to query and manipulate EDM constructs
  - Object Services:
    - services that enable you to work with the Common Language Runtime (CLR) objects in a conceptual model

- The ADO.NET Entity Framework supports:
  - Writing code against a conceptual model
  - Easy updating of applications to a different data source
  - Writing code that is independent from the storage system
  - Writing data access code that supports compile-time type-checking and syntax-checking

Using the ADO.NET Entity Data Model Tools

- Tools support:
  - Database-first design by using the Entity Data Model Wizard
    - you design and create your database before you generate your model.
    - commonly used when you are developing applications against an existing data source;
  - Code-first design by using the Generate Database Wizard
    - you design the entities for your application and then create the database structure around these entities

- ADO.NET Entity Data Model Tools includes the Entity Data Model Designer
  - for graphically creating and relating entities in a model.

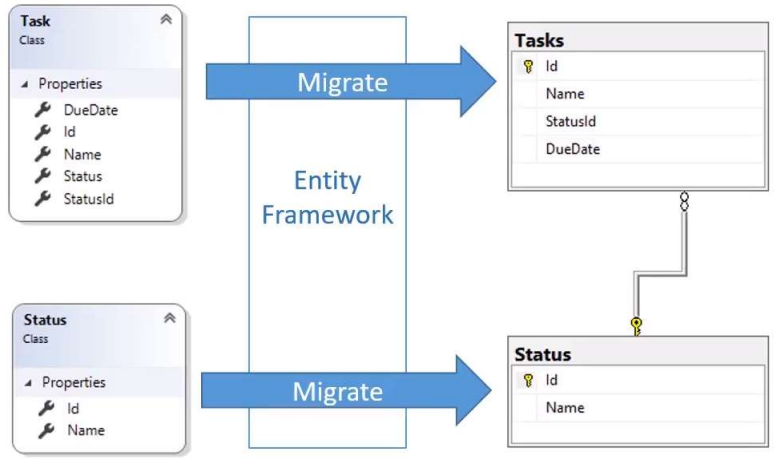
- They provide three wizards for working with models and data sources:

Wizard	Description
Entity Data Model Wizard	Enables you to generate a new conceptual model from an existing data source by using the database-first design method.
Update Model Wizard	Enables you to update an existing conceptual model with changes that are made to the data source on which it is based.
Generate Database Wizard	Enables you to generate a database from a conceptual model that you have designed in the Entity Data Model Designer by using the code-first design method.

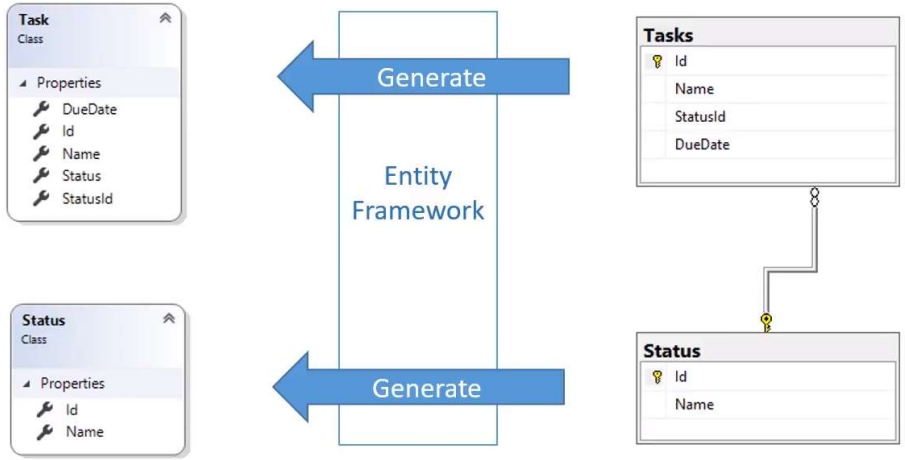
Extra

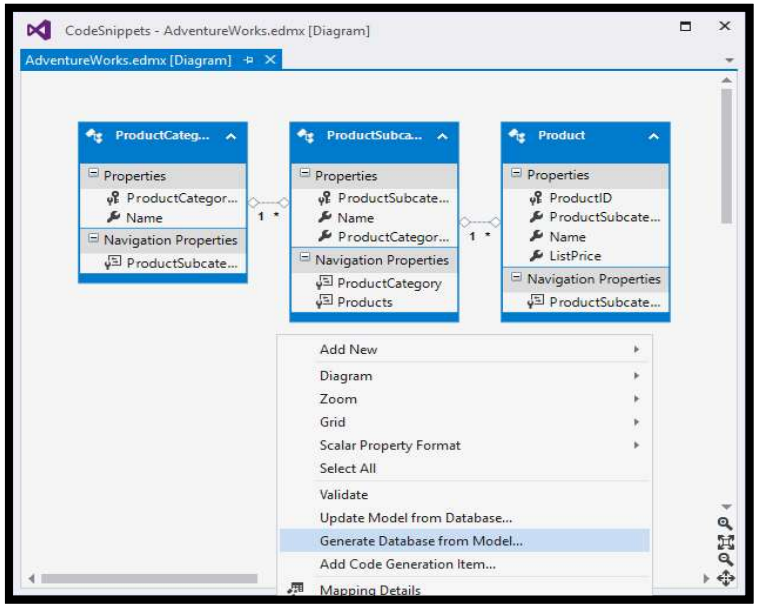
- Source: https://www.youtube.com/watch?v=S9HrLdSrVho

Code First



Database First





Customizing Generated Classes

- When you use the Entity Data Model Wizard to create a model, it automatically generates classes that expose the entities in the model to your application code.
  - These classes contain properties that provide access to the properties in the entities.

FourthCoffeeEmployeesModel.tt | FourthCoffeeEmplo...sModel.Context.cs | Employee.cs | Fourth

FourthCoffee.Employees.Employee

```
//------------------------------------------------------------------------------
// <auto-generated>
//    This code was generated from a template.
//
//    Manual changes to this file may cause unexpected behavior in your application.
//    Manual changes to this file will be overwritten if the code is regenerated.
// </auto-generated>
//------------------------------------------------------------------------------

namespace FourthCoffee.Employees
{
    using System;
    using System.Collections.Generic;

    public partial class Employee
    {
        public int EmployeeID { get; set; }
        public string FirstName { get; set; }
        public string LastName { get; set; }
        public Nullable<System.DateTime> DateOfBirth { get; set; }
        public Nullable<int> Branch { get; set; }
        public Nullable<int> JobTitle { get; set; }

        public virtual Branch Branch1 { get; set; }
        public virtual JobTitle JobTitle1 { get; set; }
    }
}
```

- Do not modify the automatically generated classes in a model
  - if at any time in the future you run the Update Model Wizard, the classes will be regenerated and your code will be overwritten.
  - the generated classes are defined as partial classes; therefore, you can extend them to add custom functionality to the classes.
  - Use partial classes and partial methods to add business functionality to the generated classes
- The following code example shows how you can add business logic to a generated class by using a partial class.

Adding Business Logic in a Partial Class

```
public partial class Employee
{
    public int GetAge()
    {
        DateTime DOB = (DateTime)_DateOfBirth;
        TimeSpan difference = DateTime.Now.Subtract(DOB);
        int ageInYears = (int)(difference.Days / 365.25);
        return ageInYears;
    }
}
```

Customizing Generated Classes

- W
  W
  ge
  in
  -
- D
  ge
  -
  -
  -
- Th
  yo
  ge

```
namespace FourthCoffee.Employees2222
{
    class Program
    {
        static void Main(string[] args)
        {
            Employee myempl = new Employee();
            myempl.SSN = "000-00-0000";
        }
    }

    public partial class Employee
    {
        public string SSN { get; set; }
    }
}
```

Reading and Modifying Data by Using the Entity Framework

- The automatically generated code files for a model also contains a partial class that inherits from the **System.Data.Entity.DbContext** class
  - **DbContext** class
    - Used for querying and working with entity data as objects.
    - It contains a default constructor which initializes the class by using the connection string that the wizard generates in the application configuration file (**App.config**)
    - contains a **DbSet** property that exposes a **DbSet(TEntity)** class for each entity in your model. The DbSet(TEntity) class represents a typed entity set that you can use to read, create, update, and delete data.

```
App.config
10  <connectionStrings>
11    <add name="FourthCoffeeEntities" connectionString="metadata=res://*/FourthCoffeeEmployeesModel.csdl|res
12  </connectionStrings>
```

```
AssemblyInfo.cs   FourthCoffeeEmplo...sModel.Context.cs   Employee.cs   FourthCoffeeEmplo...
FourthCoffee.Employees.FourthCoffeeEntities
//------------------------------------------------------------------------------
// <auto-generated>
//    This code was generated from a template.
//
//    Manual changes to this file may cause unexpected behavior in your application.
//    Manual changes to this file will be overwritten if the code is regenerated.
// </auto-generated>
//------------------------------------------------------------------------------

namespace FourthCoffee.Employees
{
    using System;
    using System.Data.Entity;
    using System.Data.Entity.Infrastructure;

    public partial class FourthCoffeeEntities : DbContext
    {
        public FourthCoffeeEntities()
            : base("name=FourthCoffeeEntities")
        {
        }

        protected override void OnModelCreating(DbModelBuilder modelBuilder)
        {
            throw new UnintentionalCodeFirstException();
        }

        public DbSet<Branch> Branches { get; set; }
        public DbSet<Employee> Employees { get; set; }
        public DbSet<JobTitle> JobTitles { get; set; }
    }
}
```

- To use the typed entity set, you create an instance of the **DbContext** class and then access the properties using the standard dot notation
  - Reading data
    - **DbSet(TEntity)** class implements the **IEnumerable** interface.
    - Hence, the **First** extension method locates the first match for the specified condition
  - Modifying data
    - you must explicitly apply changes to the data in the data source. For this call **SaveChanges** method of the **ObjectContext** object

```
FourthCoffeeEntities DBContext = new FourthCoffeeEntities();

// Print a list of employees.
foreach (FourthCoffee.Employees.Employee emp in DBContext.Employees)
{
    Console.WriteLine("{0} {1}", emp.FirstName, emp.LastName);
}
```

```
var emp = DBContext.Employees.First(e => e.LastName == "Prescott");
if (emp != null)
{
    emp.LastName = "Forsyth";
    DBContext.SaveChanges();
}
```

Lesson 2: Querying Data by Using LINQ

- Querying Data
- Demonstration: Querying Data
- Querying Data by Using Anonymous Types
- Demonstration: Querying Data by Using Anonymous Types
- Forcing Query Execution
- Demonstration: Retrieving and Modifying Grade Data Lab

Querying Data

- LINQ: an alternative to using the Entity Framework for querying data
  - supports compile-time syntax-checking and type-checking and also uses Microsoft IntelliSense in Visual Studio

- Use LINQ to query a range of data sources, including:
  - .NET Framework collections
  - SQL Server databases
  - ADO.NET data sets
  - XML documents
  - you can use it to query any data source that implements the **IEnumerable** interface
    - the syntax of the query itself does not change if you use a different type of data source

- Use LINQ to:
  - Select data
  - Filter data by row
  - Filter data by column

Querying Data

```
static FourthCoffeeEntities DBContext = new FourthCoffeeEntities();
```

- The return data type from the query below is an **IQueryable<Employee>**, enabling you to iterate through the data that is returned.

- Selecting data → → → → → → → → →→

```
IQueryable<Employee> emps = from e in DBContext.Employees
                            orderby e.LastName
                            select e;
```

- Filtering data:
  - By row (use the **where** keyword) → → → → → →

```
string _LastName = "Prescott";
IQueryable<Employee> emps = (from e in DBContext.Employees
                             where e.LastName == _LastName
                             select e);
```

  - By column (declare a **new** type in which to store a subset of columns):

```
private class FullName
{
    public string Forename { get; set; }
    public string Surname { get; set; }
}

private static void FilteringDataByColumn()
{
    IQueryable<FullName> names = from e in DBContext.Employees
                                 select new FullName { Forename = e.FirstName, Surname = e.LastName };
```

- Working with the Results
  - Use the dot notation

```
foreach (var name in names)
{
    Console.WriteLine("{0} {1}", name.Forename, name.Surname);
}
```

Querying Data

```
static FourthCoffeeEntities DBContext = new FourthCoffeeEntities();
```

```
FourthCoffeeEntities DBContext = new FourthCoffeeEntities();

IQueryable<Employee> emps = from e in DBContext.Employees
                            orderby e.LastName
                            select e;

foreach(var emp in emps)
    Console.WriteLine(emp.LastName);
```

C:\WINDOWS\system32\cmd.exe

```
Adams
Bentley
Herb
John
Kennedy
Poe
Prescott
Saylor
Press any key to continue . . .
```

Querying Data by Using Anonymous Types

- In the examples in the previous topic and demonstration, the return data was always stored in a **strongly typed IQueryable<Type>** variable;
  - You can use **anonymous types** to store the returned data by declaring the return type as an implicitly typed local variable, a **var**, **and** by using the **new** keyword in the select clause to create the instance of the type.

- Use LINQ and anonymous types to:
  - Filter data by column (see example below: the use of **var** and **new**)
  - Group data (see example below: the use of **group** clause)
  - Aggregate data (see example below: the use of **group** clause + **Count()**)
  - Navigate data (see example below: the use of **dot** notation ...)

```
IQueryable<FullName> names = from e in DBContext.Employees
                             select new FullName { Forename = e.FirstName, Surname = e.LastName };
```

```
var emps = from e in DBContext.Employees
           select new { e.FirstName, e.LastName };
```

Forcing Query Execution

- Deferred query execution—default behavior for most queries
  - a LINQ query that returns a sequence of values, is **not run until you actually try to use** some of the returned data
  - ensures that you can create a query to retrieve data in a multiple-user scenario and know that whenever it is executed **you will receive the latest information**.

- Immediate query execution—default behavior for queries that return a singleton value
  - when you define a LINQ query that returns a singleton value (for example, an **Average**, **Count**, or **Max** function), the query is run immediately
  - necessary because the query must produce a sequence to calculate the singleton result.

- Forced query execution—overrides deferred query execution:
  - You can override the default deferred query execution behavior for queries that do not produce a singleton result by calling one of the following methods on the query:
  - **ToArray**
  - **ToDictionary**
  - **ToList**

```
IList<Employee> emp = (from e in FCEntities.Employees
                       orderby e.LastName
                       select e).ToList();
```

Extra: … joined tables

Hello everyone in CSC 200,

Please find below an example of a join between two of the tables we've seen today in class

```
var emplInfos = from emp in DBContext.Employees
        join j in DBContext.JobTitles
        on emp.JobTitle equals j.JobTitleId
        select new { emp.LastName, emp.FirstName, j.Job };

foreach(var em in emplInfos)
        Console.WriteLine(em.LastName+"  "+ em.FirstName +  ",  " +em.Job);
```

And the output:

```
Adams  Terry,  Branch Manager
Poe  Toni,  Barista
Herb  Charlie,  Trainee Barista
Prescott  Diane,  Trainee Barista
John  Glen,  Branch Manager
Bentley  Sean,  Barista
Kennedy  Will,  Trainee Barista
Saylor  Dennis,  Branch Manager
```

Module Review and Takeaways

- **Question:** What advantages does LINQ provide over traditional ways of querying data?

- **Question:** Fourth Coffee wants you to add custom functionality to an existing EDM in its Coffee Sales application. You need to write a method for adding a new product to the application. In which of the following locations should you write your code?
  - ( )Option 1: In the relevant generated class in the EDM project.
  - ( )Option 2: In a partial class in the EDM project.