Entity Framework

EF Core Database First

Lesson Objectives

- EF Database First
- Entity-Table Mapping
- Create context from database
- DbContext, DbSet
- Entity Relationship
- Basic of LINQ
- Querying in Entity Framework

Section 1

EF DATABASE FIRST

In previous lesson

- *Why do you choose EF Database First?*

Edmx file in Entity Framework

- *.edmx is basically an XML file which is generated when we added Entity Framework model.
- It is Entity Data Model Xml which contains designer (Model) and code file(.cs).

Three Major schema of EDM

- CSDL (Conceptual Schema Definition Language)
  - ✓ stores the schema of cs file which generated for each table of database.
- SSDL (Storage Schema Definition Language)
  - ✓ stores the schema of database which we created
- MSL (Mapping Specification Language)
  - ✓ stores the mapping of SSDL and CSDL.

Entity-Table Mapping

- Each entity in EDM is mapped with the database table.
- Column of the table is mapped to Property of the Entity
- Table name, column name, data type are generated automatically, based on table schema.

Session 2

CREATE CONTEXT FROM DATABASE

Create context from database

EF Core does not support visual designer for DB model and wizard to create the entity and context classes similar to EF 6. So, we need to do reverse engineering using the Scaffold-DbContext command.

Step 1: Create entity and context classes for the database
Step 2: Use Scaffold-DbContext to create a model based on your existing database

Scaffold-DbContext Command

The following parameters can be specified with Scaffold-DbContext in Package Manager Console:

```
Scaffold-DbContext [-Connection] [-Provider] [-OutputDir] [-Context] [-Schemas>] [-Tables>]
                   [-DataAnnotations] [-Force] [-Project] [-StartupProject] [<CommonParameters>]
```

Scaffold-DbContext Command

In Visual Studio, select menu Tools -> NuGet Package Manger -> Package Manger Console and run the following command:

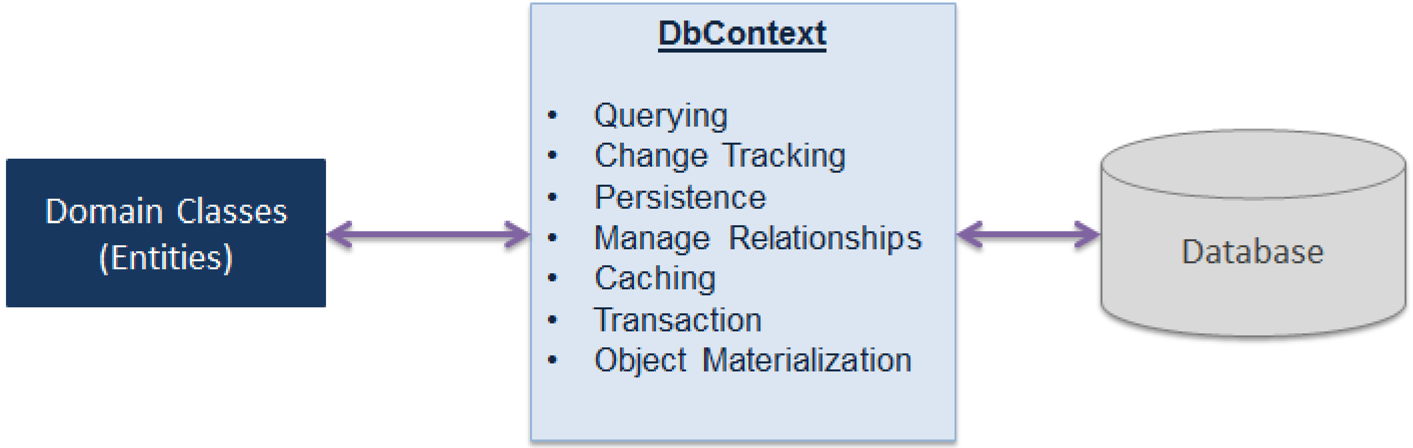
```
PM> Scaffold-DbContext "Server=.\SQLExpress;Database=DBName;Trusted_Connection=True;" Microsoft.EntityFrameworkCore.SqlServer -OutputDir Models
```

Section 3

DBCONTEXT

DbContext

- A bridge between domain or entity classes and the database.



DbContext Activities

- **Querying**
  - ✓ Converts LINQ-to-Entities queries to SQL query and sends them to the database.
- **Change Tracking**
  - ✓ Keeps track of changes that occurred on the entities after querying from the database.
- **Persisting Data**
  - ✓ Performs the Insert, Update and Delete operations to the database, based on entity states.

DbContext Activities

- **Caching**
  - ✓ Provides first level caching by default.
  - ✓ It stores the entities which have been retrieved during the life time of a context class.
- **Manage Relationship**
  - ✓ Manages relationships using CSDL, MSL and SSDL.
- **Object Materialization**
  - ✓ Converts raw data from the database into entity objects.

DbContext Methods

- Entry
  - ✓ Gets an DbEntityEntry for the given entity.
  - ✓ The entry provides access to change tracking information and operations for the entity.
- SaveChanges
  - ✓ Executes INSERT, UPDATE and DELETE commands to the database for the entities with Added, Modified and Deleted state.
- SaveChangesAsync
  - ✓ Asynchronous method of SaveChanges()

DbContext Methods

- Set
  - ✓ Creates a DbSet<TEntity> that can be used to query and save instances of TEntity.
- OnModelCreating
  - ✓ Override this method to further configure the model that was discovered by convention from the entity types exposed in DbSet<TEntity> properties on your derived context.

DbContext Properties

- ChangeTracker
  - ✓ Provides access to information and operations for entity instances that this context is tracking.
- Configuration
  - ✓ Provides access to configuration options.
- Database
  - ✓ Provides access to database related information and operations.

Section 4

DBSET

DbSet

- Represents an entity set that can be used for Create, Read, Update, and Delete operations. (CRUD)
- Map to database table or view.

DbSet Methods

- Add
  - ✓ Input: entity object
  - ✓ Output: added entity
  - ✓ Description: Adds the given entity to the context with the Added state. When the changes are saved, the entities in the Added states are inserted into the database. After the changes are saved, the object state changes to Unchanged.
  - ✓ Example: dbcontext.Students.Add(studentEntity)

DbSet Methods

- Create
  - ✓ Input: Non
  - ✓ Output: entity object
  - ✓ Description: Creates a new instance of an entity for the type of this set. This instance is not added or attached to the set. The instance returned will be a proxy if the underlying context is configured to create proxies and the entity type meets the requirements for creating a proxy.
  - ✓ Example: var newStudentEntity = dbcontext.Students.Create();

DbSet Methods

- Remove
  - ✓ Input: entity object
  - ✓ Output: entity object
  - ✓ Description: Marks the given entity as Deleted. When the changes are saved, the entity is deleted from the database. The entity must exist in the context in some other state before this method is called.
  - ✓ Example: dbcontext.Students.Remove(studentEntity);

DbSet Methods

- Find
  - ✓ Input: params object[] keyValues
  - ✓ Output: entity object
  - ✓ Description: Uses the primary key value to find an entity tracked by the context. If the entity is not in the context, then a query will be executed and evaluated against the data in the data source, and null is returned if the entity is not found in the context or in the data source. Note that the Find also returns entities that have been added to the context but have not yet been saved to the database.
  - ✓ Example: Student studEntity = dbcontext.Students.Find(studentId);

DbSet Methods

- Attach
  - ✓ Input: entity object
  - ✓ Output: entity which was passed as parameter
  - ✓ Description: Attaches the given entity to the context in the Unchanged state.
  - ✓ Example: dbcontext.Students.Attach(studentEntity);

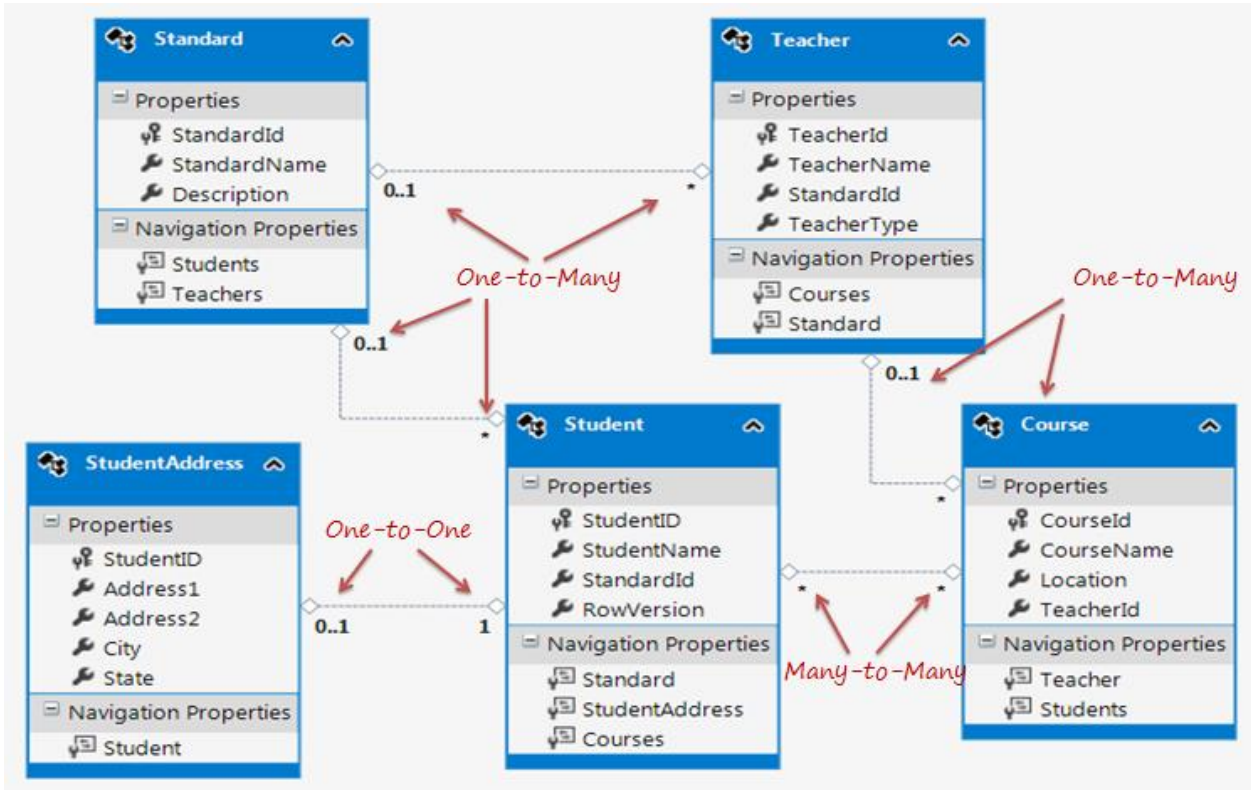
Section 5

ENTITY RELATIONSHIP

Entity Relationship

- Entity framework supports three types of relationships
  - ✓ One-to-Many
  - ✓ One-to-One
  - ✓ Many-to-Many

Entity Relationship

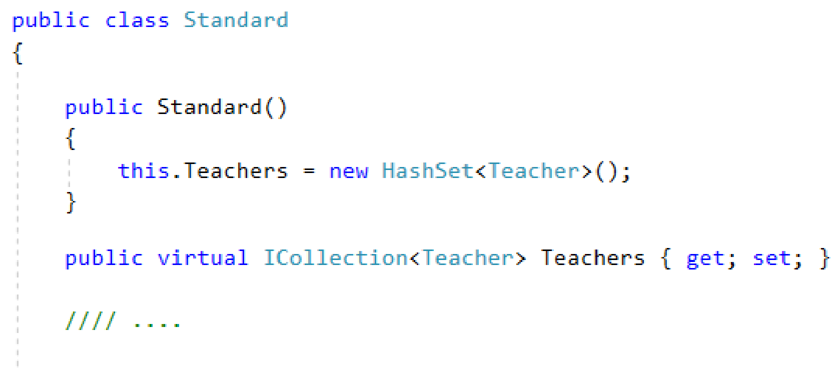


One-to-Many Relationship

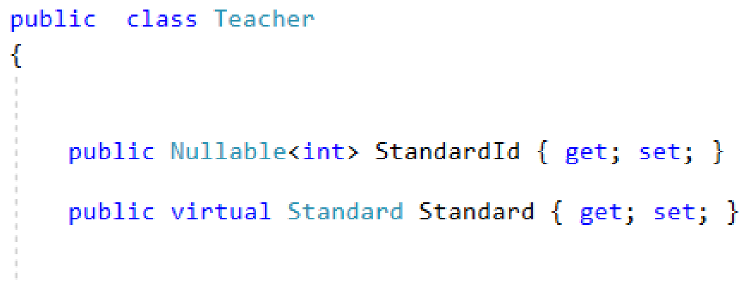
- One ***parent*** can have many ***children*** whereas a ***child*** can associate with only one ***parent***
  - ✓ *Example: The **Standard** and **Teacher** entities have a One-to-Many relationship*
    - *Standard can have many Teachers*
    - *Teacher can associate with only one Standard*

One-to-Many Relationship

- The parent entity has the collection navigation property children
- The child entity has a parent navigation property
- The child entity also contains the foreign key to parent primary key



```
public class Standard
{
    public Standard()
    {
        this.Teachers = new HashSet<Teacher>();
    }

    public virtual ICollection<Teacher> Teachers { get; set; }

    //// ....
```



```
public  class Teacher
{

    public Nullable<int> StandardId { get; set; }

    public virtual Standard Standard { get; set; }
```

One-to-One Relationship

- One ***entity*** can have one or zero ***entity*** *in relationship*
  - ✓ *Example: One department has one director*
  - ✓ *Example: A student can have only one or zero addresses*

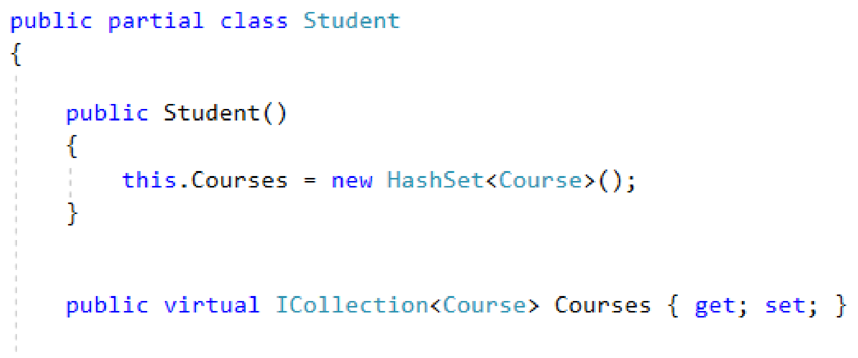
```
public partial class Student
{
    public virtual StudentAddress StudentAddress { get; set; }
    //// ....

public partial class StudentAddress
{

    public int StudentID { get; set; }

    public virtual Student Student { get; set; }
```

Many-to-Many Relationship

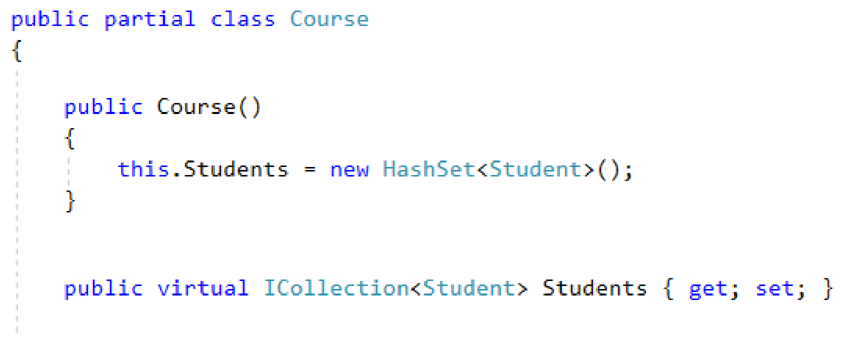
- One Product can appear in many Orders, and one Order can have many Products
- One Student can enroll for many Courses and also, one Course can be taught to many Students.

Many-to-Many Relationship

- In SQL: break many-to-many relationship to 2 one-to-many relationships
- Joining table only includes PKs of 2 tables
- In EF: use 2 collection navigation properties for 2 entities



```
public partial class Student
{

    public Student()
    {
        this.Courses = new HashSet<Course>();
    }


    public virtual ICollection<Course> Courses { get; set; }
```



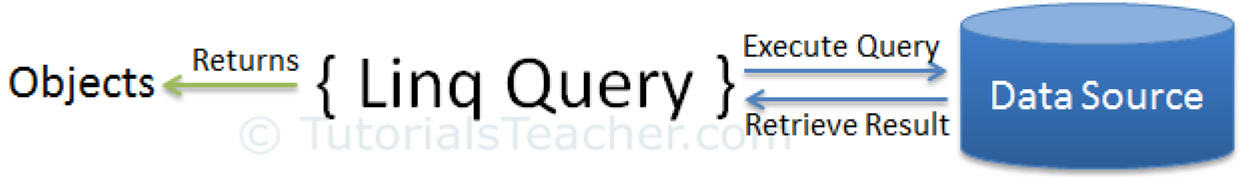
```
public partial class Course
{

    public Course()
    {
        this.Students = new HashSet<Student>();
    }


    public virtual ICollection<Student> Students { get; set; }
```

Section 6

BASIC OF LINQ

LINQ

- LINQ (Language Integrated Query) is uniform query syntax in C# and VB.NET to retrieve data from different sources and formats.
- LINQ queries return results as objects. It enables you to uses object-oriented approach on the result set and not to worry about transforming difference formats of results into objects.



Advantages of LINQ

- **Familiar language:** Developers don’t have to learn a new query language for each type of data source or data format.
- **Less coding:** It reduces the amount of code to be written as compared with a more traditional approach.

Advantages of LINQ

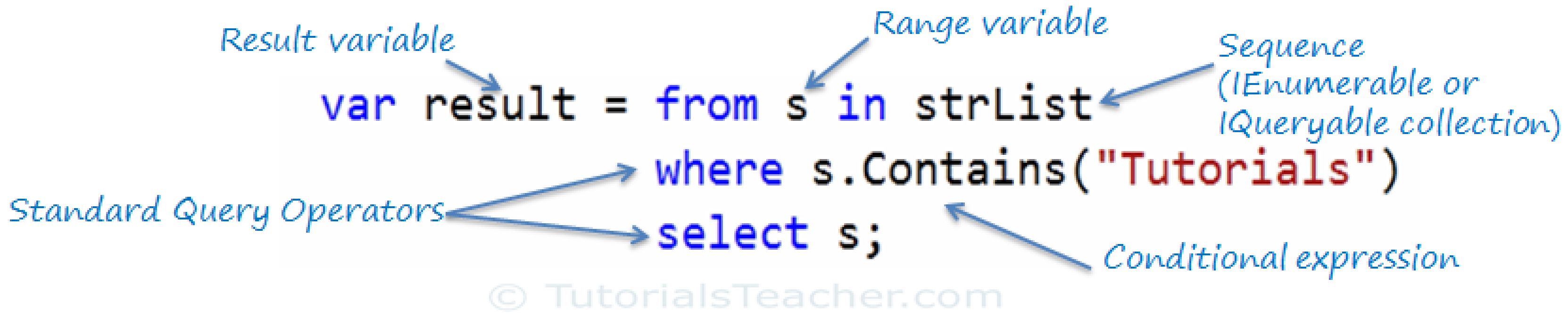
- **Readable code:** LINQ makes the code more readable so other developers can easily understand and maintain it.
- **Standardized way of querying multiple data sources:** The same LINQ syntax can be used to query multiple data sources.

Advantages of LINQ

- **Compile time safety of queries:** It provides type checking of objects at compile time.
- **IntelliSense Support:** LINQ provides IntelliSense for generic collections.
- **Shaping data:** You can retrieve data in different shapes.

LINQ Query Syntax

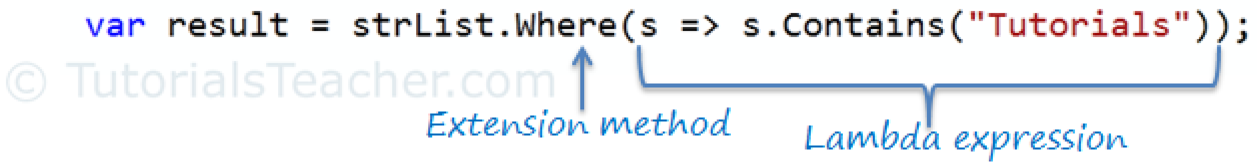
- Query syntax is similar to SQL (Structured Query Language) for the database.
- It is defined within the C# or VB code.



```
var result = from s in strList
             where s.Contains("Tutorials")
             select s;
```

LINQ Method Syntax

- Method syntax (also known as fluent syntax) uses extension methods included in the Enumerable or Queryable static class,
- It is similar to call the extension method of any class.



```
var result = strList.Where(s => s.Contains("Tutorials"));
```

Standard Query Operators

- Where
- OrderBy/ThenBy
- OrderByDescending/ThenByDescending
- All/Any
- Contains
- …

Standard Query Operators

- Average/Count/Max/Min/Sum/Distinct
- Skip/SkipWhile
- Take/TakeWhile
- ToArray/ToList/ToDictionary
- … and more

Section 7

QUERYING IN ENTITY FRAMEWORK

LINQ-to-Entities Query

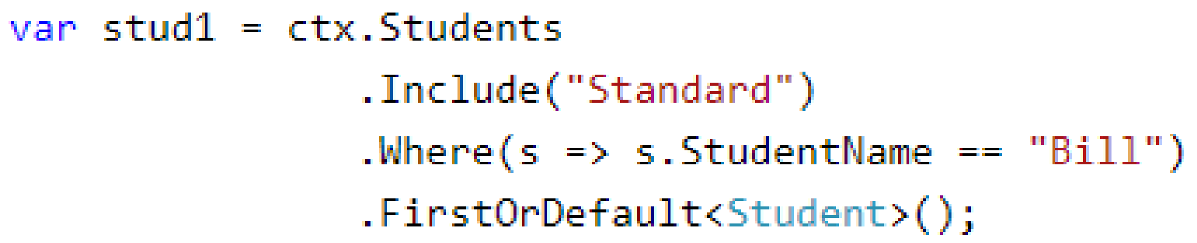
- Use LINQ for querying against DbSet. It will be converted to an SQL query.
- EF API executes this SQL query
  - ✓ to the underlying database,
  - ✓ gets the flat result set,
  - ✓ converts it into appropriate entity objects and
  - ✓ returns it as a query result

Eager Loading

- Is the process whereby a query for one type of entity also loads related entities as part of the query,
- Don't need to execute a separate query for related entities.
- Eager loading is achieved using the **Include()** method.

Eager Loading

- Example: Gets all the students from the database along with its standards using the Include() method.



```
var stud1 = ctx.Students
               .Include("Standard")
               .Where(s => s.StudentName == "Bill")
               .FirstOrDefault<Student>();
```

Lazy Loading

- Is delaying the loading of related data, until specifically request for it.
- Navigation property should be defined as public, virtual.
- Context will **NOT** do lazy loading if the property is not defined as virtual.

Lazy loading with proxies

- The simplest way to use lazy-loading is by installing the Proxies package

```
PM> Install-Package Microsoft.EntityFrameworkCore.Proxies -Version 6.0.8
```

- After enabling it with a call to UseLazyLoadingProxies.

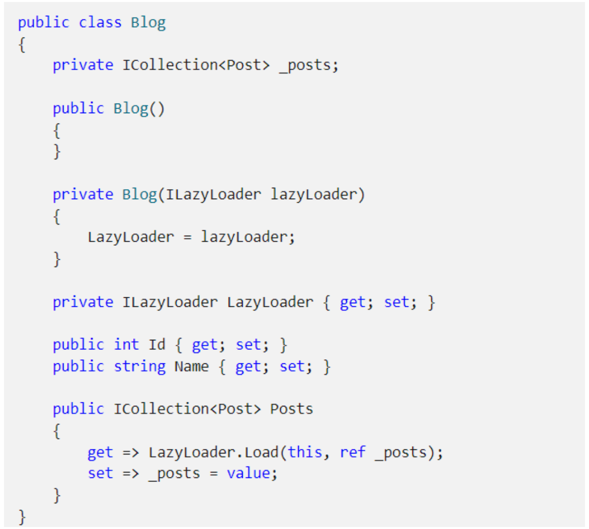
For example:

```
protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
    => optionsBuilder
        .UseLazyLoadingProxies()
        .UseSqlServer(myConnectionString);
```

Lazy loading without proxies

- Lazy-loading without proxies work by injecting the ILazyLoader service into an entity, as described in Entity Type Constructors.

For example:



```
public class Blog
{
    private ICollection<Post> _posts;

    public Blog()
    {
    }

    private Blog(ILazyLoader lazyLoader)
    {
        LazyLoader = lazyLoader;
    }

    private ILazyLoader LazyLoader { get; set; }

    public int Id { get; set; }
    public string Name { get; set; }

    public ICollection<Post> Posts
    {
        get => LazyLoader.Load(this, ref _posts);
        set => _posts = value;
    }
}
```

Explicit Loading

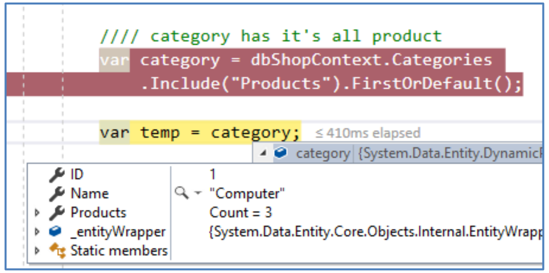
- **To** load the entities when lazy loading is disabled
- **By** calling the Load method for the related entities.
  - ✓ Reference: to load single navigation property
  - ✓ Collection: to load collections

```
var category = dbShopContext.Categories.Find(1);
dbShopContext.Entry(category).Collection(p => p.Products).Load();
```

Comparison

Eager Loading

Always load for first time



```
//// category has it's all product
var category = dbShopContext.Categories
        .Include("Products").FirstOrDefault();

var temp = category;  ≤ 410ms elapsed
```

```
category  {System.Data.Entity.Dynamic
ID                  1
Name                "Computer"
Products            Count = 3
_entityWrapper      {System.Data.Entity.Core.Objects.Internal.EntityWrapp
Static members
```

Lazy Loading

Load for first time requested

```
//// Does not load products at this time
var category = dbShopContext.Categories
    .FirstOrDefault();

//// Working without products
var temp1 = category;

//// Load products to be used
var products = category.Products;

//// Working with products
var temp2 = category;
```

Explicit Loading

Load when explicit call

```
//// When lazy loading is disabled
//// Products are not loaded at this time
var category = dbShopContext.Categories
    .FirstOrDefault();

//// Working without products
var temp1 = category;

//// Explicit loading products to be used
dbShopContext.Entry(category)
    .Collection(p => p.Products).Load();

//// Working with products
var temp2 = category;
```

When to use what

- Use Eager Loading when the relations are not too much. Thus, Eager Loading is a good practice to reduce further queries on the Server.
- Use Eager Loading when you are sure that you will be using related entities with the main entity everywhere.
- Use Lazy Loading when you are using one-to-many collections.

When to use what

- Use Lazy Loading when you are sure that you are not using related entities instantly.
- When you have turned off Lazy Loading, use Explicit loading when you are not sure whether or not you will be using an entity beforehand.

Summary

- Use EF Database First when you have existing database, with important data
- Use Entity Data Model file to create models and context
- DbContext is a bridge between domain or entity classes and the database

Summary

- DbSet is used to manipulate data
- Entity Framework uses LINQ for querying data
- 3 types: Eager Loading vs Lazy Loading vs Explicit Loading

Fpt Software CAMPUS

Thank you