Entity Framework

Database-First

Lesson Objectives

- EF Database First
- Entity-Table Mapping
- DbContext, DbSet
- Entity Relationship

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.

Practice

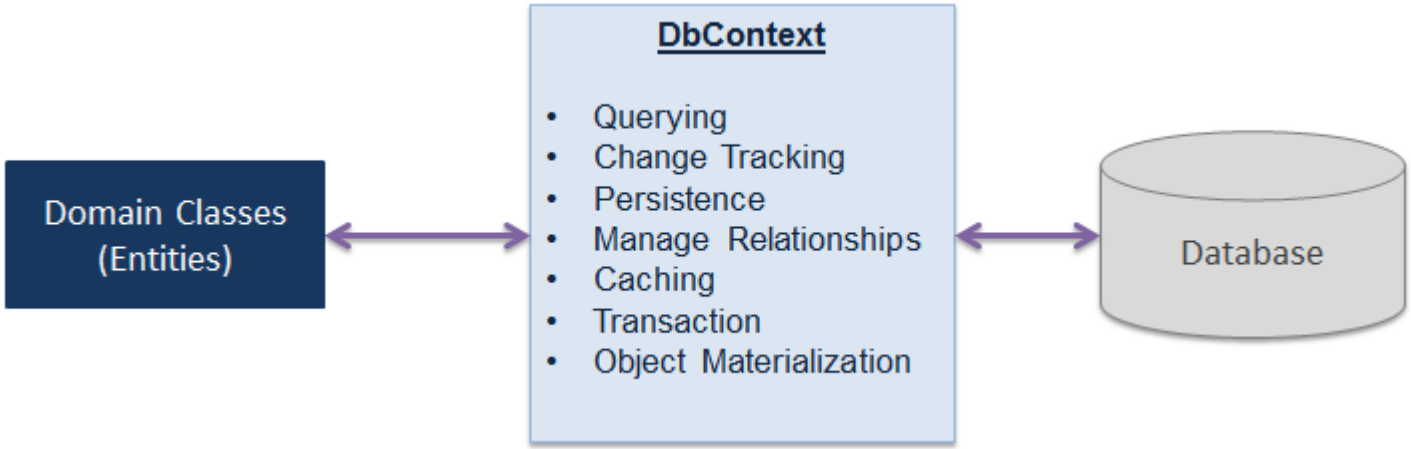
- Create database with one table
- Create project then install Entity Framework package
- Create Entity Data Model and check mapping
- Update SQL table, update EDM

Section 2

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.

Practice

- Explorer DbContext from previous project
- Find corresponding SQL Connection String

Section 3

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);

Practice

- Add item Student then check in SQL
- Get added Student
- Update Student in both (connected and disconnected) scenarios
- Delete Student

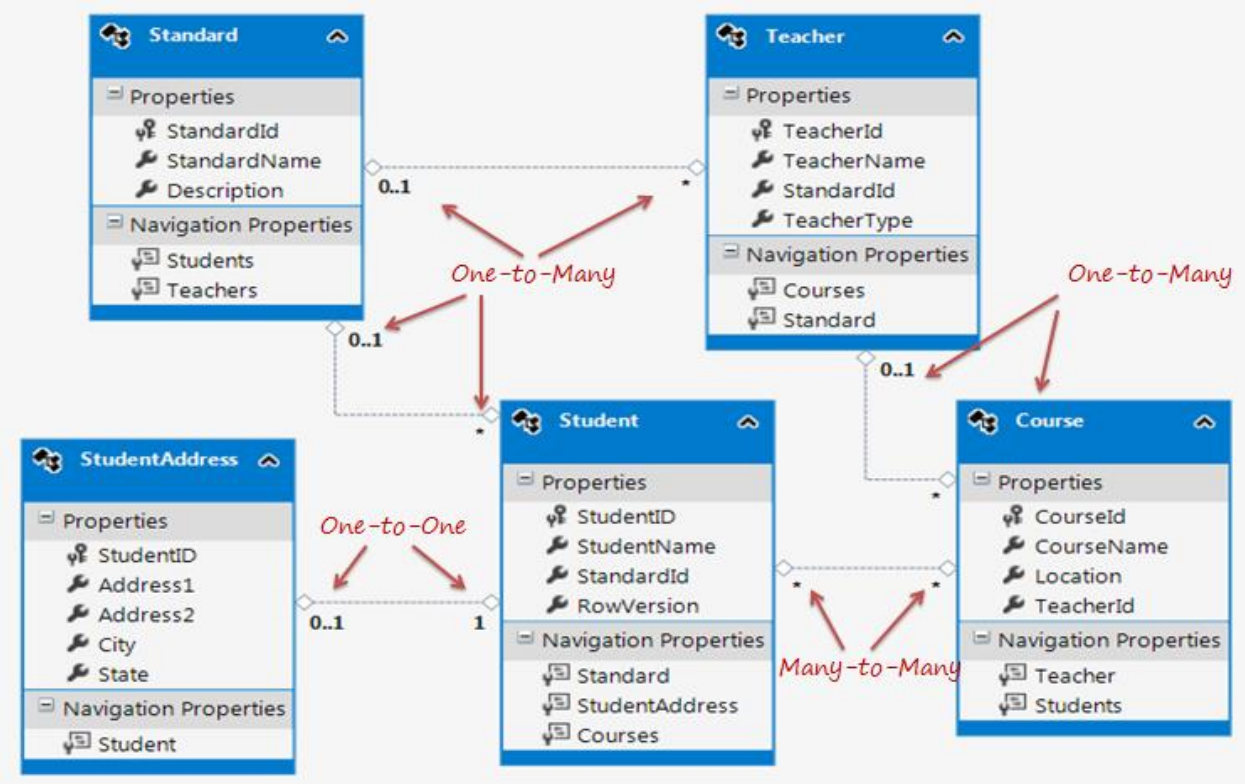
Section 4

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; }
}
```

Practice

- Add new SQL table then set one-to-many relationship
- Update Entity Data Model
- Practice to CRUD data in the relationship

Thank you