Microsoft®
.NET

UNIT 8:
ENTITY FRAMEWORK (EF)

CONTENTS

- What is EF?
- What is ORM?
- Setup EF
- Approaches
- Query with EDM
- Loading in EF

What is Entity Framework

- **From Microsoft:**

The Microsoft ADO.NET Entity Framework is an Object/Relational Mapping (ORM) framework that enables developers to work with relational data as domain-specific objects, eliminating the need for most of the data access plumbing code that developers usually need to write.

Using the Entity Framework, developers issue queries using LINQ, then retrieve and manipulate data as strongly typed objects. The Entity Framework's ORM implementation provides services like change tracking, identity resolution, lazy loading, and query translation so that developers can focus on their application-specific business logic rather than the data access fundamentals.

What is ORM

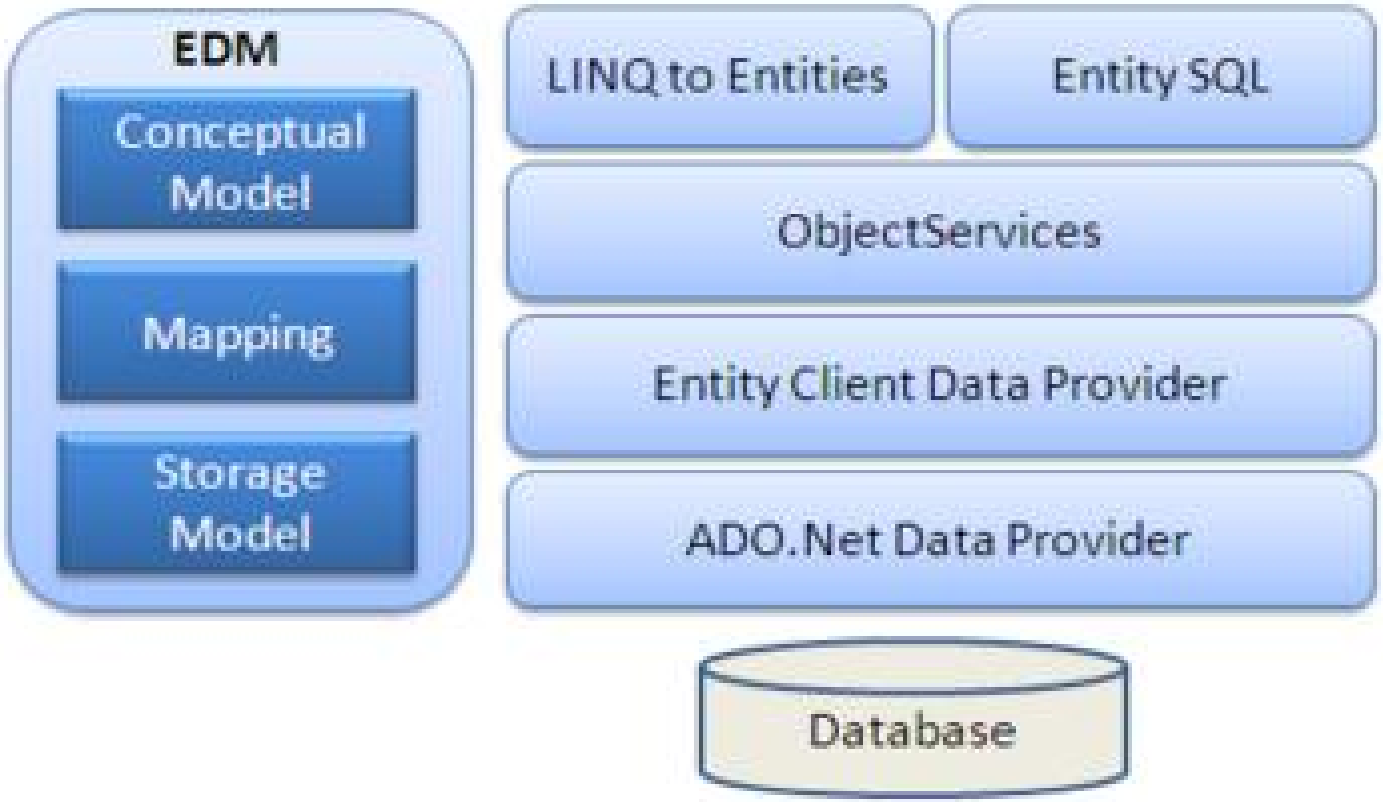
- **ORM is a tool for storing data from domain objects to relational database.**

What is ORM

- ORM is a tool for storing data from domain objects to relational database.
- O/RM includes three main parts: Domain class objects, Relational database objects and Mapping information on how domain objects map to relational database objects (tables, views & storedprocedures)
- ORM allows us to keep our database design separate from our domain class design.
- It also automates standard CRUD operation

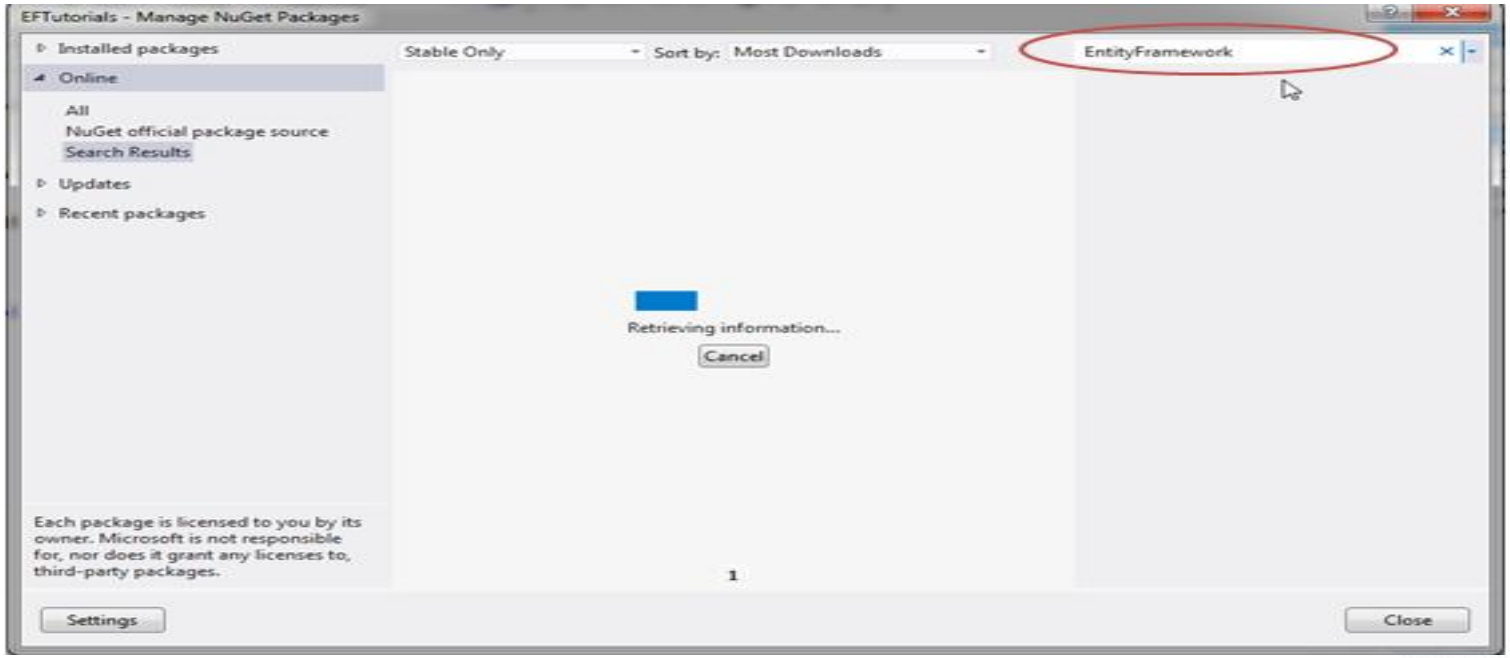
Entity Framework Architecture

- **Architect of EF:**
  - **EDM (Entity Data Model)**



Setup EF

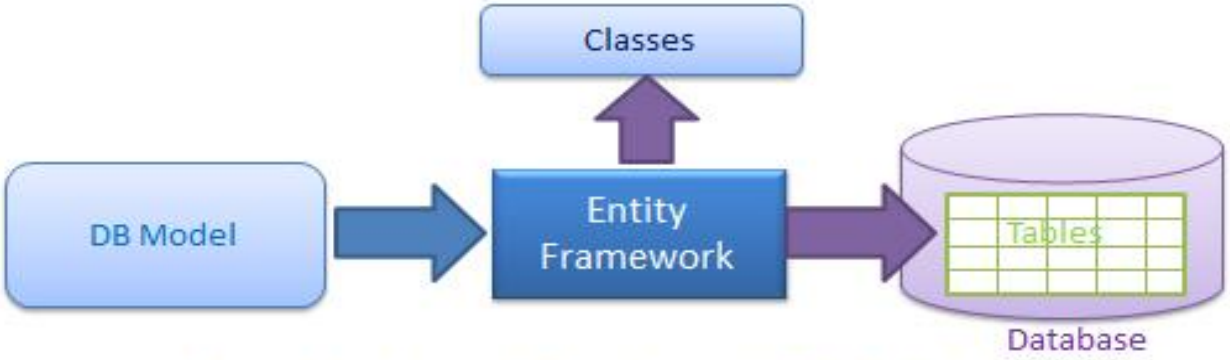
- **Install EF via Nuget:**
  - **Right click on project, select Manage Nuget Packages**
  - **Then select Online in left bar and search for EntityFramework**
  - **Select EntityFramework and click on Install.**



Approaches

Generate Data Access Classes for Existing Database

Create Database from the Domain Classes



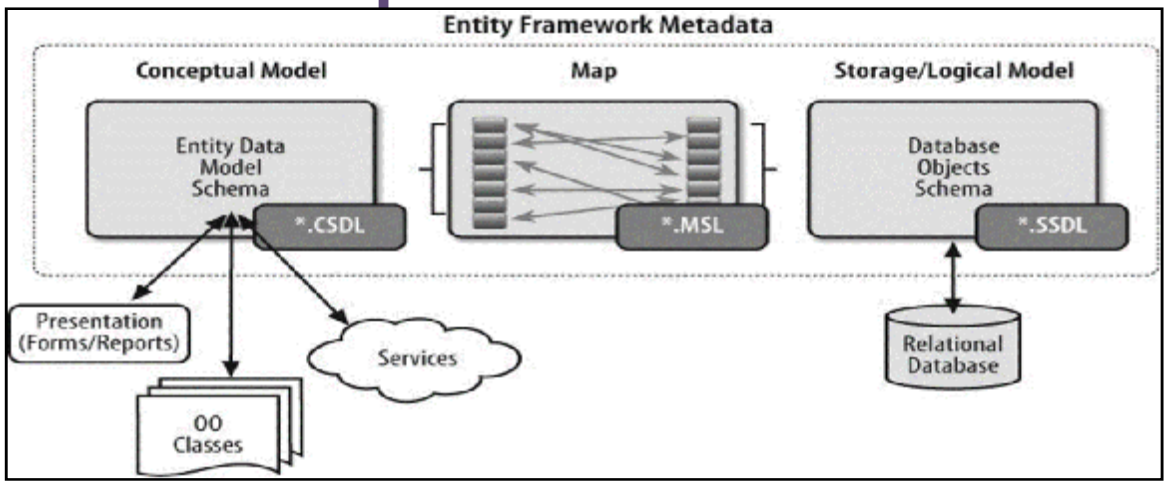
Create Database and Classes from the DB Model design

Create Entity Data Model

- Entity Data Model (EDM) is a model that describes entities and the relationships between them.
- Add EDM
- Entity-Table Mapping
- DBContext: is an important part of Entity Framework. It is a bridge between your domain or entity classes and the database.

Create Entity Data Model

- **The output of the designer is an XML document that consists of 3 parts:**




```
<?xml version="1.0" encoding="utf-8"?>
<edmx:Edmx Version="2.0" xmlns:edmx="http://schemas.microsoft.com/ado/2008/10/edmx">
  <!-- EF Runtime content -->
  <edmx:Runtime>
    <!-- SSDL content -->
    <edmx:StorageModels>...</edmx:StorageModels>
    <!-- CSDL content -->
    <edmx:ConceptualModels>...</edmx:ConceptualModels>
    <!-- C-S mapping content -->
    <edmx:Mappings>...</edmx:Mappings>
  </edmx:Runtime>
  <!-- EF Designer content (DO NOT EDIT MANUALLY BELOW HERE) -->
  <Designer xmlns="http://schemas.">...</Designer>
</edmx:Edmx>
```

Entity Relationships

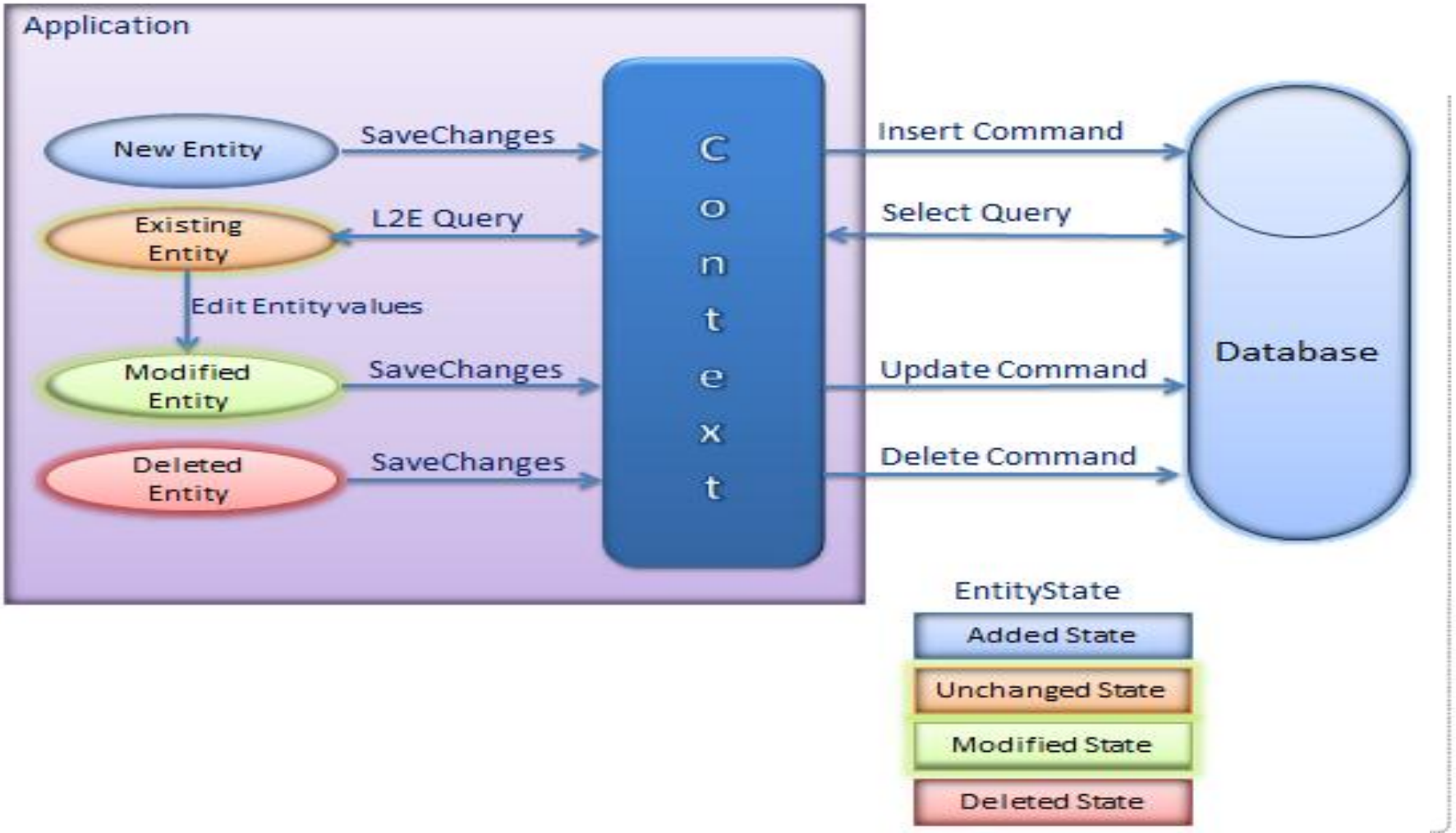
- Entity can have two types of properties, Scalar and Navigation properties.
- Entity framework supports three types of relationships, same as database:

1) One to One
2) One to Many
3) Many to Many.

Entity Lifecycle

- During an entity's lifetime, each entity has an entity state based on the operation performed on it via the context (DbContext).
- The entity state is an enum of type *System.Data.Entity.EntityState* that includes the following values:
  - Added
  - Deleted
  - Modified
  - Unchanged
  - Detached

Entity Lifecycle



Query with EDM

- **Entity framework supports three types of queries:**
  - **LINQ to Entities: operates on entity framework entities to access the data from the underlying database. You can use LINQ method syntax or query syntax when querying with EDM**
  - **Entity SQL: It is processed by the Entity Framework's Object Services directly. It returns ObjectQuery instead of IQueryable. You need ObjectContext to create a query using Entity SQL**
  - **Native SQL:ou can execute native SQL queries for a relational database**

Query with EDM

- **Linq-to-Entities Projection Queries:**
  - **First/FirstOrDefault**
  - **Single/SingleOrDefault**
  - **ToList**
  - **GroupBy**
  - **OrderBy**

```
using (var ctx = new SchoolDBEntities())
{
    var student = (from s in ctx.Students
                   where s.StudentName == "Student1"
                   select s).FirstOrDefault<Student>();
}
```

DbClass

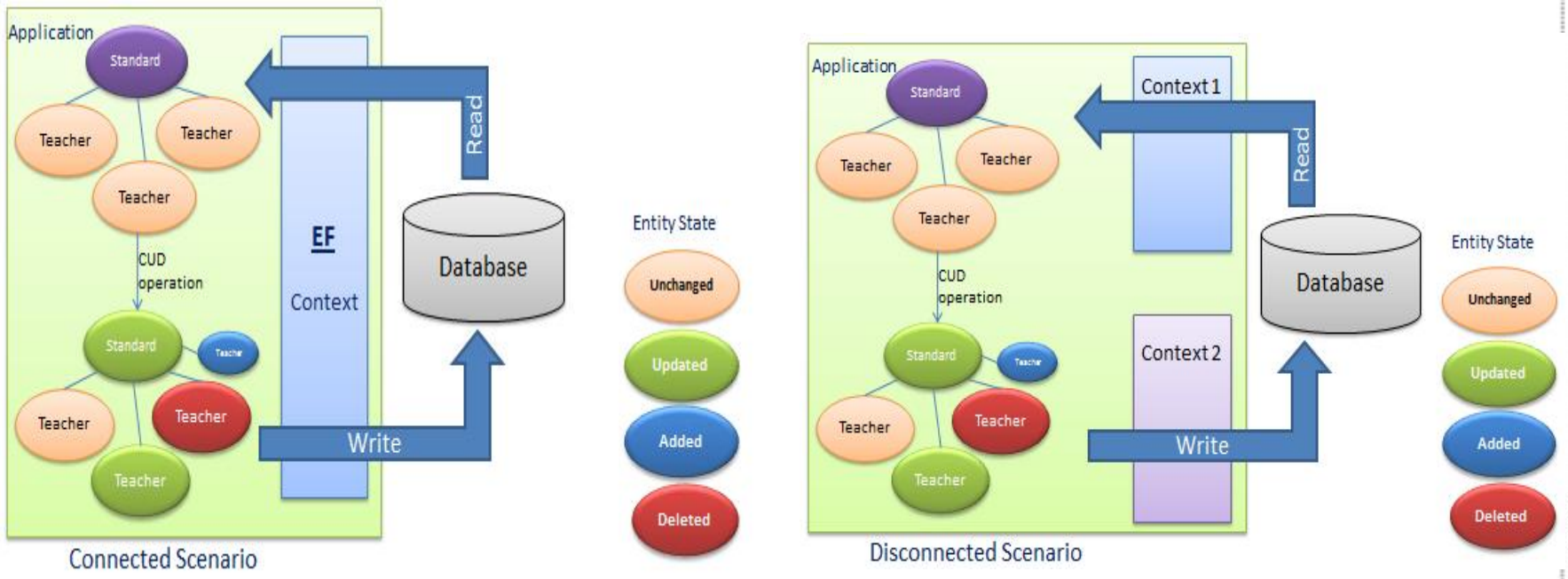
- DBSet class represents an entity set that is used for CRUD operations.
- A generic version of DBSet (DbSet<TEntity>) can be used when the type of entity is not known at build time.
- Some of the important methods of DBSet class:
  - Add
  - Attach(Entity)
  - Create
  - Find(int)
  - Include
  - Remove

DBEntityEntry Class

- DBEntityEntry is an important class, which is useful in retrieving various information about an entity.
- You can get an instance of DBEntityEntry of a particular entity by using Entry method of DBContext
- DBEntityEntry enables you to access entity state, current, and original values of all the property of a given entity
- DbEntityEntry enables you to set Added, Modified or Deleted EntityState to an entity

CRUD in EF

- **There are two scenarios when persisting an entity using EF:**



CRUD in EF

- **CRUD in connected scenario:**
  - **Context will automatically detect the changes and update the state of an entity.**

```
using (var context = new SchoolDBEntities())
{
    var studentList = context.Students.ToList<Student>();
    //Perform create operation
    context.Students.Add(new Student() {
              StudentName = "New Student" });
    //Perform Update operation
    Student studentToUpdate =studentList.Where(s => s.StudentName
== "student1").FirstOrDefault<Student>();
    studentToUpdate.StudentName = "Edited student1";
    //Perform delete operation
    context.Students.Remove(studentList.ElementAt<Student>(0));
    //Execute Inser, Update & Delete queries in the database
    context.SaveChanges();
}
```

CRUD in EF

- **CRUD in connected scenario**
  - **Attach entities with the new context instance**
  - **Set appropriate EntityStates to these entities manually**
- **Use some methods:**
  - **DbSet.Add()**
  - **DbSet.Attach()**
  - **DbContext.Entry()**

```
//disconnected entity graph
Student disconnectedStudent = new Student() { StudentName = "New Student" };
disconnectedStudent.StudentAddress = new StudentAddress() { Address1 = "Address", City = "City1" };

using (var ctx = new SchoolDBEntities())
{
    //add disconnected Student entity graph to new context instance - ctx
    ctx.Students.Add(disconnectedStudent);

    // get DbEntityEntry instance to check the EntityState of specified entity
    var studentEntry = ctx.Entry(disconnectedStudent);
    var addressEntry = ctx.Entry(disconnectedStudent.StudentAddress);

    Console.WriteLine("Student EntityState: {0}",studentEntry.State);

    Console.WriteLine("StudentAddress EntityState: {0}",addressEntry.State);
}
```

Eager Loading

- Eager loading is the process whereby a query for one type of entity also loads related entities as part of the query.
- Eager loading is achieved using the Include method of Iqueryable

```
using (var context = new SchoolDBEntities())
{
    var res = (from s in context.Students.Include("Standard")
               where s.StudentName == "Student1"
               select s).FirstOrDefault<Student>();
}
```

- Load multiple levels of related entities:

```
using (var ctx = new SchoolDBEntities())
    {
        stud = ctx.Students.Include("Standard.Teachers")
                           .Where(s => s.StudentName == "Student1")
                           .FirstOrDefault<Student>();
    }
```

Lazy loading

- One of the important functions of Entity Framework is lazy loading.
- Lazy loading means delaying the loading of related data, until you specifically request for it

```
using (var ctx = new SchoolDBEntities())
    {
        //Loading students only
        IList<Student> studList = ctx.Students.ToList<Student>();

        Student std = studList[0];

        //Loads Student address for particular Student only (seperate SQL query)
        StudentAddress add = std.StudentAddress;
    }
```

- Rules for lazy loading:
  - *context.Configuration.ProxyCreationEnabled* should be true.
  - *context.Configuration.LazyLoadingEnabled* should be true.
  - Navigation property should be defined as public, virtual. Context will NOT do lazy loading if the property is not defined as virtual.

Explicit Loading

- **Use the Load method of DBEntityEntry object**

```
using (var context = new SchoolDBEntities())
{
    //Disable Lazy loading
    context.Configuration.LazyLoadingEnabled = false;

    var student = (from s in context.Students
                   where s.StudentName == "Bill"
                   select s).FirstOrDefault<Student>();

    context.Entry(student).Reference(s => s.Standard).Load();
}
```

```
using (var context = new SchoolDBEntities())
{
    context.Configuration.LazyLoadingEnabled = false;

    var student = (from s in context.Students
                   where s.StudentName == "Bill"
                   select s).FirstOrDefault<Student>();

    context.Entry(student).Collection(s => s.Courses).Load();
}
```