Connected And Disconnected Scenarios in Entity Framework

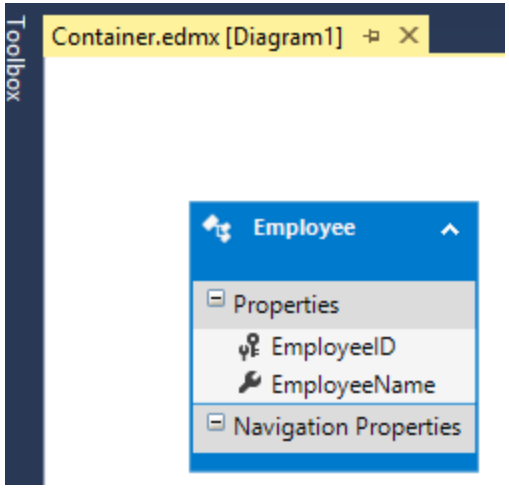
Disconnected Mode

Let's suppose we want to display some data in a Presentation Layer and we are using some n-tier application, so it would be better to open the context, fetch the data and finally close the context. Since here we have fetched the data and closed the context the entities that we have fetched are no longer tracked and this is the disconnected scenario.

Connected Mode

For a connected scenario let us suppose we have a Windows service and we are doing some business operations with that entity so we will open the context, loop through all the entities, do our business operations and then save the changes with the same context that we opened in the beginning. In the case of a connected scenario the changes are tracked by the context but in the case of a disconnected scenario we need to inform the context about the state of the entity.

Table Employee



Connected Mode

```
public class EmployeeRepository
{
    /// <summary>
    /// This Function will change employee Name
    /// </summary>
    public void ChangeEmployeeName()
    {
        try
        {
            using(var context = new ContainerEntities())        Line 1
            {
                var employeeList = context.Employees.ToList();        Line2

                //Loop through employeeList
                foreach(var emp in employeeList)
                {
                    emp.EmployeeName = "Edited" + emp.EmployeeName;          Line3
                }

                context.SaveChanges();          Line4
            }
        }
        catch(Exception ex)
        {
            //TODO Log Exception
        }
```

Connected Mode

- **Line 1** : We opened the context.
- **Line 2**: We got all the employees in the database.
- **Line 3**: We are editing the employee name.
- **Line 4**: We are saving the changes to the database.

Again, since we have opened the context and we are editing the entities, the context is tracking the changes so we need not provide the change state explicitly here.

Disconnected Mode

```
/// <summary> ...
public bool UpdateEmployee(Employee emp)
{
    try
    {
        using (var context = new ContainerEntities())
        {
            context.Employees.Attach(emp);
            context.Entry(emp).State = System.Data.Entity.EntityState.Modified;
            context.SaveChanges();
        }
    }
    catch (Exception ex)
    {
        //TODO Log Exception
    }

    return false;
```

Line1

Line 2

Line 3

Line 4

Disconnected Mode

- **Line 1** : We have opened the context.
- **Line 2** : We have attached the disconnected entity to the context.
- **Line 3** : We need to explicitly define the state.
- **Line 4** : Finally go ahead and save the changes. (This will update the entity.)

Since we are discussing the entity state here, there can be five entity states for very entity and depending upon which Entity Framework decides the operation when save changes is called.

Disconnected Mode

```
/// <summary>
///
/// </summary>
/// <param name="emp"></param>
/// <returns></returns>
public bool DeleteEmployee(Employee emp)
{
    try
    {
        using (var context = new ContainerEntities())    Line 1
        {
            context.Employees.Attach(emp);        Line 2
            context.Entry(emp).State = System.Data.Entity.EntityState.Deleted; Line 3
            context.SaveChanges();            Line 4
        }
    }
    catch (Exception ex)
    {
        //TODO Log Exception
    }

    return false;
}
```

Disconnected Mode

- **Line 1**: We have opened the context.
- **Line 2**: We have attached the disconnected entity to the context.
- **Line 3**: We need to explicitly define the state.
- **Line 4**: Finally go ahead and save the changes. (This will delete the entity.)

Lazy Loading

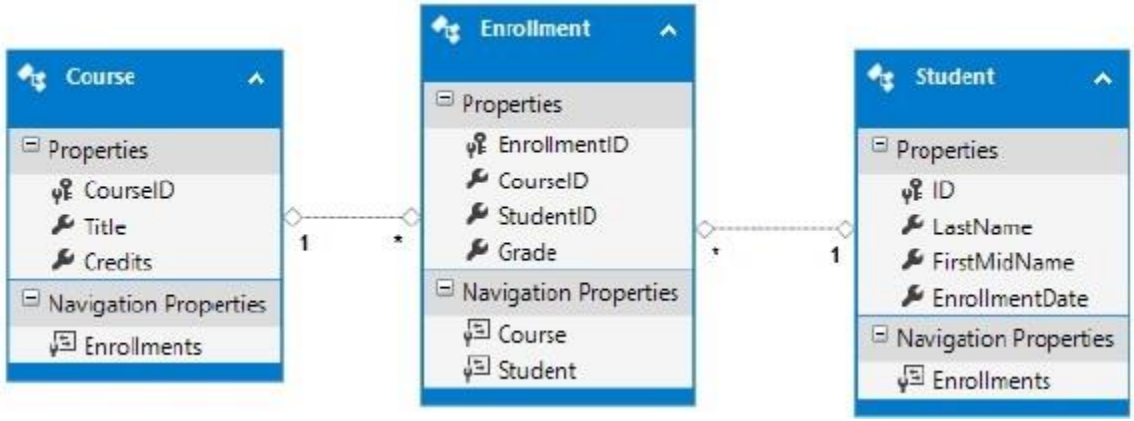
Lazy loading is the process whereby an entity or collection of entities is automatically loaded from the database the first time that a property referring to the entity/entities is accessed. Lazy loading means delaying the loading of related data, until you specifically request for it.

- When using POCO entity types, lazy loading is achieved by creating instances of derived proxy types and then overriding virtual properties to add the loading hook.
- Lazy loading is pretty much the default.

Lazy Loading

- If you leave the default configuration, and don't explicitly tell Entity Framework in your query that you want something other than lazy loading, then lazy loading is what you will get.
- For example, when using the Student entity class, the related Enrollments will be loaded the first time the Enrollments navigation property is accessed.
- Navigation property should be defined as public, virtual. Context will **NOT** do lazy loading if the property is not defined as virtual.

Lazy Loading



Lazy Loading

Following is a Student class which contains navigation property of Enrollments.

```
public partial class Student {

  public Student() {
    this.Enrollments = new HashSet<Enrollment>();
  }

  public int ID { get; set; }
  public string LastName { get; set; }
  public string FirstMidName { get; set; }
  public System.DateTime EnrollmentDate { get; set; }

  public virtual ICollection<Enrollment> Enrollments { get; set; }
}
```

Lazy Loading

Let’s take a look into a simple example in which student list is loaded from the database first and then it will load the enrollments of a particular student whenever you need it.

Lazy Loading

```
class Program {
  static void Main(string[] args) {
    using (var context = new UniContextEntities()) {
      //Loading students only
      IList<Student> students = context.Students.ToList<Student>();
      foreach (var student in students) {
        string name = student.FirstMidName + " " + student.LastName;
        Console.WriteLine("ID: {0}, Name: {1}", student.ID, name);
        foreach (var enrollment in student.Enrollments) {
          Console.WriteLine("Enrollment ID: {0}, Course ID: {1}",
            enrollment.EnrollmentID, enrollment.CourseID);
        }
      }
      Console.ReadKey();
    }
  }
}
```

Lazy Loading

ID: 1, Name: Ali Alexander

Enrollment ID: 1, Course ID: 1050

Enrollment ID: 2, Course ID: 4022

Enrollment ID: 3, Course ID: 4041

ID: 2, Name: Meredith Alonso

Enrollment ID: 4, Course ID: 1045

Enrollment ID: 5, Course ID: 3141

Enrollment ID: 6, Course ID: 2021

ID: 3, Name: Arturo Anand

Enrollment ID: 7, Course ID: 1050

ID: 4, Name: Gytis Barzdukas

Enrollment ID: 8, Course ID: 1050

Enrollment ID: 9, Course ID: 4022

ID: 5, Name: Yan Li

Enrollment ID: 10, Course ID: 4041

ID: 6, Name: Peggy Justice

Enrollment ID: 11, Course ID: 1045

ID: 7, Name: Laura Norman

Enrollment ID: 12, Course ID: 3141

Turn Off Lazy Loading

Turning Off for Specific Navigation Properties

Lazy loading of the Enrollments collection can be turned off by making the Enrollments property non-virtual as shown in the following example.

```
public partial class Student {
  public Student() {
    this.Enrollments = new HashSet<Enrollment>();
  }
  public int ID { get; set; }
  public string LastName { get; set; }
  public string FirstMidName { get; set; }
  public System.DateTime EnrollmentDate { get; set; }

  public ICollection<Enrollment> Enrollments { get; set; }
}
```

Turn Off Lazy Loading

Turn Off for All Entities

Lazy loading can be turned off for all entities in the context by setting a flag on the Configuration property to false as shown in the following example.

```
public partial class UniContextEntities : DbContext {

  public UniContextEntities(): base("name = UniContextEntities") {
    this.Configuration.LazyLoadingEnabled = false;
  }

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

Turn Off Lazy Loading

After turning off lazy loading, now when you run the above example again you will see that the Enrollments are not loaded and only student data is retrieved.

ID: 1, Name: Ali Alexander

ID: 2, Name: Meredith Alons

ID: 3, Name: Arturo Anand

ID: 4, Name: Gytis Barzduka

ID: 5, Name: Yan Li

ID: 6, Name: Peggy Justice

ID: 7, Name: Laura Norman

ID: 8, Name: Nino Olivetto

Explicit Loading

- When you disabled the lazy loading, it is still possible to lazily load related entities, but it must be done with an explicit call.
- Let's take a look at the following example in which lazy loading is disabled and then the student whose first name is Ali is retrieved.
- Student information is then written on console. If you look at the code, enrollments information is also written but Enrollments entity is not loaded yet so foreach loop will not be executed.
- After that Enrollments entity is loaded explicitly now student information and enrollments information will be written on the console window.

Explicit Loading

```
class Program {
  static void Main(string[] args) {
    using (var context = new UniContextEntities())
    {
      context.Configuration.LazyLoadingEnabled = false;
            var student = (from s in context.Students where s.FirstMidName ==
        "Ali" select s).FirstOrDefault<Student>();
      string name = student.FirstMidName + " " + student.LastName;
      Console.WriteLine("ID: {0}, Name: {1}", student.ID, name);
      foreach (var enrollment in student.Enrollments)
      {
        Console.WriteLine("Enrollment ID: {0}, Course ID: {1}",
          enrollment.EnrollmentID, enrollment.CourseID);

      }
      Console.WriteLine();
```

Explicit Loading

```
Console.WriteLine("Explicitly loaded Enrollments");
        Console.WriteLine();
        context.Entry(student).Collection(s ⇒ s.Enrollments).Load();
        Console.WriteLine("ID: {0}, Name: {1}", student.ID, name);
        foreach (var enrollment in student.Enrollments) {
          Console.WriteLine("Enrollment ID: {0}, Course ID: {1}",
             enrollment.EnrollmentID, enrollment.CourseID);
        }
        Console.ReadKey();
    }
  }
}
```

Explicit Loading

When the above example is executed, you will receive the following output. First only student information is displayed and after explicitly loading enrollments entity, both student and his related enrollments information is displayed.

ID: 1, Name: Ali Alexander

Explicitly loaded Enrollments

ID: 1, Name: Ali Alexander

Enrollment ID: 1, Course ID: 1050

Enrollment ID: 2, Course ID: 4022

Enrollment ID: 3, Course ID: 4041

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 by the use of the **Include method**.

For example, when querying students, eager-load their enrollments. The students and their enrollments will be retrieved in a single query.

Let’s take a look at the following example in which all the students with their respective enrollments are retrieved from the database by using eager loading.

Eager Loading

```
class Program {

  static void Main(string[] args) {
    using (var context = new UniContextEntities()) {
      // Load all students and related enrollments
      var students = context.Students
        .Include(s ⇒ s.Enrollments).ToList();
      foreach (var student in students) {
        string name = student.FirstMidName + " " + student.LastName;
        Console.WriteLine("ID: {0}, Name: {1}", student.ID, name);
        foreach (var enrollment in student.Enrollments) {
          Console.WriteLine("Enrollment ID: {0}, Course ID: {1}",
            enrollment.EnrollmentID, enrollment.CourseID);
        }
      }
      Console.ReadKey();
    }
  }
}
```

Eager Loading

When the above code is compiled and executed you will receive the following output.

ID: 1, Name: Ali Alexander
    Enrollment ID: 1, Course ID: 1050
    Enrollment ID: 2, Course ID: 4022
    Enrollment ID: 3, Course ID: 4041
ID: 2, Name: Meredith Alonso
    Enrollment ID: 4, Course ID: 1045
    Enrollment ID: 5, Course ID: 3141
    Enrollment ID: 6, Course ID: 2021
ID: 3, Name: Arturo Anand
    Enrollment ID: 7, Course ID: 1050
ID: 4, Name: Gytis Barzdukas
    Enrollment ID: 8, Course ID: 1050
    Enrollment ID: 9, Course ID: 4022