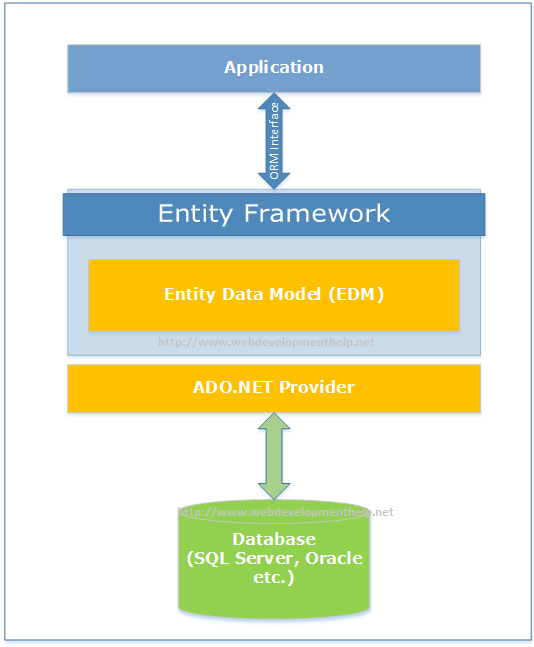
**Q. What is Entity Framework?**

**ANS**:

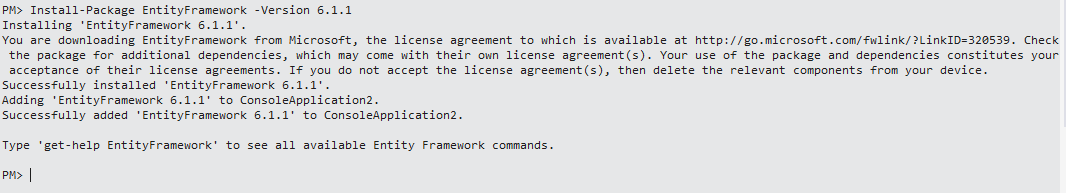
Microsoft Entity Framework (EF) is an Object Relational Mapping framework that provides an abstraction layer (a kind of mapping) between two incompatible systems (i.e. Object oriented programming languages with Relational databases). It enables us to interact with relational data in an object oriented way, meaning we can retrieve and manipulate data as strongly typed objects using LINQ queries.



### Q. What is the Nuget Package console command to install Entity Framework, if it’s not there in you project at that time?

### ANS:

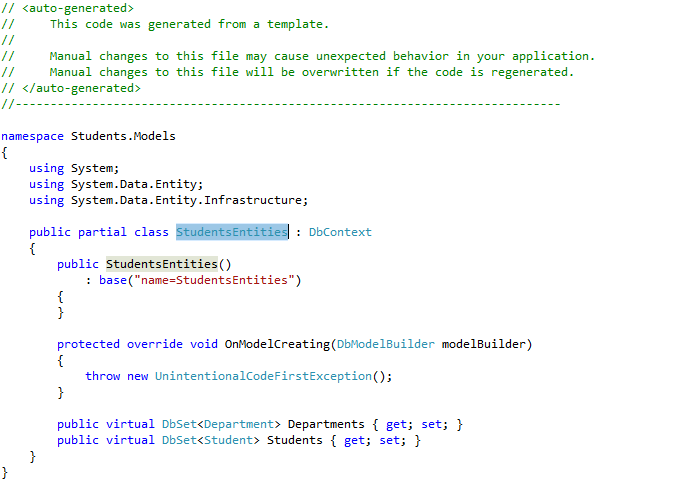
***Install-Package EntityFramework -Version 6.1.1***



### Q. What’s new in Entity Framework 6?

**ANS**:

* Customizing Code First Conventions.
* Logging of database commands.
* Stored Procedure Mapping.
* Asynchronous Queries and Save support.



With EF6, in applications with lots of tables and relationships defined, our context objects open faster. We should also be better insulated from dropped connections (at least, if the drop is transient and not permanent — nothing is going to help there).  EF6 also generates SQL faster from LINQ queries than before (though it’s the same SQL that’s being generated as in earlier versions of EF, so your actual data access won’t be any faster or slower).

### Q. Can we access a model in Entity Framework without primary key defined?

### ANS:

No, but we can access data.

Using Primary Key:

*Student student = db.Students.Find(id);*

Without Primary Key:

*Student student = db.Students.Where(x=>x.StudentName==”Ahmad”);*

### Q. Explain what does .edmx file contains?

### ANS:

.edmx file is an XML file, which declares a conceptual model, a storage model and the mapping between these models.  This file also consists the information that is used by ADO.NET entity data model designer to render a model graphically. It consists of all the mapping details of how object maps with SQL tables.  It is divided into three categories SSDL, CSDL, and MSL.

EDMX (Entity Data Model XML) is an XML file which contains all the mapping details of how your objects map with SQL tables. The EDMX file is further divided into three sections: CSDL, SSDL, and MSL.

### Q. Mention what is CSDL, SSDL and MSL sections in an EDMX file?

### ANS:

* **CSDL**: It stands for Conceptual Schema Definition Language, it is the conceptual abstraction which is exposed to the application
* **SSDL**: It stands for Storage Schema Definition Language, it defines the mapping with our RDBMS data structure
* **MSL**: It stands for Mapping Schema Language, it connects the SSDL and CSDL

### https://www.codeproject.com/KB/database/676309/a4.jpg

### Q. Explain difference between LINQ to SQL and ENTITY FRAMEWORK

### ANS:

|  |  |  |
| --- | --- | --- |
| SR No | LINQ to SQL | ENTITY FRAMEWORK |
| 1 | It works only with SQL Server Database | It works with various database like DB2, MYSQL, SQL Server etc. |
| 2 | To maintain the relation it generates a .dbml | It creates an .edmx files initially and relation is maintained using 3 different files .msl, .csdl and .ssdl |
| 3 | It cannot generate database from model | It can generate database from model |
| 4 | It permits one to one mapping between the entity classes and relational views/tables | Between the entity classes and relational tables, it permits one-to-one, one-to-many and many-to-many |
| 5 | It enables you to query data using DataContext | It enables you to query data using EntitySQL, DBContext, and ObjectContext |
| 6 | It provides tightly coupled approach | It provides loosely coupled approach |

### Q. Explain approaches in Entity Framework.

### ANS:

There are following three approaches in Entity Framework –

1. Model First

2. Database First

3. Code First

**1. Model First**

In this approach, model classes and their relation are created first using the ORM designer and the physical database will be generated using this model. The Model First approach means we create a diagram of the entity and relation that will be converted automatically into a code model.

**2. Database First**

The Database First approach enables us to create an entity model from the existing database. This approach helps us to reduce the amount of code that we need to write. The following procedure will create an entity model using the Database First approach.

**2. Code First**

The Code First approach enables us to create a model and their relation using classes and then create the database from these classes. It enables us to work with the Entity Framework in an object-oriented manner. Here we need not worry about the database structure.

## Q. People say Entity Framework runs slow, why?

## ANS:

## By default EF has lazy loading behaviour. Due to this default behaviour if you are loading a large number of records and especially if they have foreign key relationships, you can have performance issues. So you need to be cautious if you really need lazy loading behaviour for all scenarios. For better performance, disable lazy loading when you are loading a large number of records or use stored procedures.

## Q. Can you explain lazy loading in a detailed manner?

## ANS:

Lazy loading is a concept where we load objects on demand rather than loading everything in one go. Consider a situation where you have 1 to many relationships between the Customer and Address objects. Now let’s say you are browsing the customer data but you do not want address data to be loaded at that moment. But the time you start accessing the address object you would like to load address data from the database.

Entity Framework has lazy loading behaviour by default enabled. For instance, consider the below code. When we are doing for each on the Customer object, the Address object is not loaded. But the time you start doing for each on the address collection, the Address object is loaded from SQL Server by firing SQL queries.

So in simple words, it will fire a separate query for each address record of the customer, which is definitely not good for a large number of records.

Hide   Copy Code

MyEntities context = new MyEntities();

var Customers = context.Customers.ToList();

foreach (Customercust in Customers) *// In this line no address object loaded*

{

foreach(Address add in cust.Addresses){}*// Address object is loaded here*

}

## Q. How can we turn off lazy loading?

## ANS:

The opposite of lazy loading is eager loading. In eager loading we load the objects beforehand. So the first thing is we need to disable lazy loading by setting LazyLoadingEnabled to false.

Hide   Copy Code

contextContextOptions.LazyLoadingEnabled = false;

Now we have to explicitly tell EF what objects we want to load by using the include function. Below is a simple sample code where we tell EF to load customer as well as address objects by using the include function.

Now the customer object and the related address objects will be loaded in one query rather than multiple queries.

var employees = context.Customers.Include("Addresses").Take(5);

## Q. What are POCO classes in Entity Framework?

## ANS:

POCO means Plain Old C# Object. When EDMX creates classes, they are cluttered with a lot of entity tags. For instance, below is a simple customer class generated using Entity Framework. Many times we would like to use simple .NET classes and integrate them with Entity Framework.

Entity Framework allows this. In other words you can create a simple .NET class and use the entity context object to load your simple .NET classes.

Below is a simple class generated by EF which is cluttered with a lot of EF attributes.

Hide   Copy Code

[EdmEntityTypeAttribute(NamespaceName="CustomermytestModel", Name="Customer")]

[Serializable()]

[DataContractAttribute(IsReference=true)]

public partial class Customer : EntityObject

{

#region Factory Method

/// *<summary>*

/// *Create a new Customer object.*

/// *</summary>*

/// *<param name="id" />Initial value of the Id property.*

/// *<param name="customerCode" />Initial value of the CustomerCode property.*

/// *<param name="customername" />Initial value of the Customername property.*

public static Customer CreateCustomer(global::System.Int32 id,

global::System.String customerCode, global::System.String customername)

{

Customer customer = new Customer();

customer.Id = id;

customer.CustomerCode = customerCode;

customer.Customername = customername;

return customer;

}

#endregion

#region Primitive Properties

## Q. How do we implement POCO in Entity Framework?

## ANS:

To implement POCO is a three step process:

* Go to the designer and set the code generation strategy to NONE. This step means that you would be generating the classes on your own rather than relying on EF auto code generation.
* Now that we have stopped the auto generation of code, we need to create the domain classes manually. Add a class file and create the domain classes like the Customer class we created.

public class Customer

{

private string \_customerName;

public string CustomerName

{

get { return \_customerName; }

set { \_customerName = value; }

}

private int \_Customerid;

public int Customerid

{

get { return \_Customerid; }

set { \_Customerid = value; }

}

}

* Write your Context layer code inheriting from ObjectContext. This code you can copy paste from the behind code of EF, also before disabling auto-generation.

public partial class Test123Entities : ObjectContext

{

public Test123Entities()

: base("name=Test123Entities", "Test123Entities")

{

this.ContextOptions.LazyLoadingEnabled = true;

OnContextCreated();

}

partial void OnContextCreated();

public ObjectSet<Customer> Customers

{

get

{

if ((\_Customers == null))

{

\_Customers = base.CreateObjectSet<Customer>("Customers");

}

return \_Customers;

}

}

private ObjectSet<Customer> \_Customers;

public void AddToCustomers(Customer customer)

{

base.AddObject("Customers", customer);

}

}

And finally you can use the above code in your client as if you were using EF normally.

Test123Entities oContext = new Test123Entities();

List<Customer> oCustomers = oContext.Customers.ToList<Customer>();

## Q. In POCO classes do we need EDMX files?

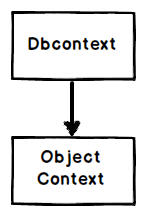
## ANS:

## Yes, you will still need EDMX files because the context object reads the EDMX files to do the mapping.

## Q. What is the difference between DbContext and ObjectContext?

## ANS:

DbContext is a wrapper around ObjectContext, it’s a simplified version of ObjectContext.



As a developer you can start with DbContext as it’s simple to use. When you feel that some of the operations cannot be achieved by DbContext, you can then access ObjectContext from DbContext, as shown in the below code:

Hide   Copy Code

((IObjectContextAdapter)dbContext).ObjectContext

## Q. How can we read records using Entity Framework classes?

## ANS:

In order to browse through records you can create the object of the context class and inside the context class you will get the records.

For instance, in the below code snippet we are looping through a customer object collection. This customer collection is the output given by the context class CustomermytextEntities.

CustomermytestEntities obj = new CustomermytestEntities();

foreach (Customer objCust in obj.Customers)

{}

## Q. How can we add, update, and delete using EF?

## ANS:

Create the object of your entity class, add it to the data context using AddObject method, and then call the SaveChanges method.

Hide   Copy Code

CustomermytestEntities obj = new CustomermytestEntities();

Customer objCust = new Customer();

objCust.CustomerCode = "1001";

obj.Customers.AddObject(objCust);

obj.SaveChanges();

If you want to update, select the object, make changes to the object, and call AcceptAllChanges.

Hide   Copy Code

CustomermytestEntities objContext = new CustomermytestEntities();

Customer objCustomer = (Customer)objContext.Customers.FirstOrDefault();

objCustomer.CountryCode = "NEP";

objContext.AcceptAllChanges();

If you want to delete, call the DeleteObject method as shown in the below code snippet:

Hide   Copy Code

CustomermytestEntities objContext = new CustomermytestEntities();

Customer objCustomer = (Customer)objContext.Customers.FirstOrDefault();

objContext.DeleteObject(objCustomer);