12/16/2015

Entity Framework

Advanced Databases

Suela Isaj & Moditha Hewasinghage
IT4BI

Contents

Introduction

A database is a group of data organized in a meaningful way. In the most traditional way of thinking is a group of linked tables. However we should not be that conservative in our way of thinking. A group of data can be organized in a lot of creative ways, more conceptually and less technically. An entity could be a tuple, a row in a spreadsheet, an object or even all of them, having the ability to transform its shape depending on the circumstances.

While working with applications and programming, accessing a database is frequently needed. It looks like the program and the database stored in some SQL Server or other provided speak different languages and it needs external translation continuously. The programmers need to be bilingual and also fast to translate all the time. Therefore a need for an intermediate layer is fundamental to proceed.

Entity Framework is the bridge that links naturally both sides. It is an Object Relational Mapping Framework. Object Relational Mapping is basically a translation of traditional databases into programming elements, objects. Entity Framework goes beyond this capability, it offers a stable environment to map a relational database quite efficiently. Entity Frameworks load the data, tables, relationships automatically from the actual database, no need of previously declaring objects and variables. Querying comes naturally too. Nevertheless, it is not only a translation of databases into codes. It goes even further, it can create a database from scratch, by declaring and running a conceptual object oriented model. Therefore, the motivation to explore it becomes strong!

1. Entity Framework Features

1.1Entities

Entities in Entity Framework have some characteristics that will be introduced as follows (Klein, 2010):

Like objects:

- Entities have a known type.
- Entities have properties, and these properties can hold scalar values.
- Entity properties can hold references to other entities.
- Each entity has a distinct identity.

Differ from objects:

- Entities live within a collection.
- Each entity has associations with other entities.
- Entities have primary keys that uniquely identify the entity.

Like relational data:

- Entities live within an entity set.
- Entities have relationships to other entities.

- They have a primary key.

Differ from relational data:

- Entities support complex types.
- Entities support inheritance.
- Entities do not have physical storage knowledge.

Obviously from the comparison between objects and relational data, entities in Entity Framework are more powerful since they are hybrids, bringing the advantages and eliminating disadvantages of both.

1.2 Entity Framework Architecture

Entity Framework provides a user friendly Graphical User Interface for interacting with the database, files will be generated automatically and they are EDM (Entity Data Model) format. EDM file has the extension .edmx and contains the conceptual, storage and mapping components. The components of Entity Framework are clearly shown by the following schema:

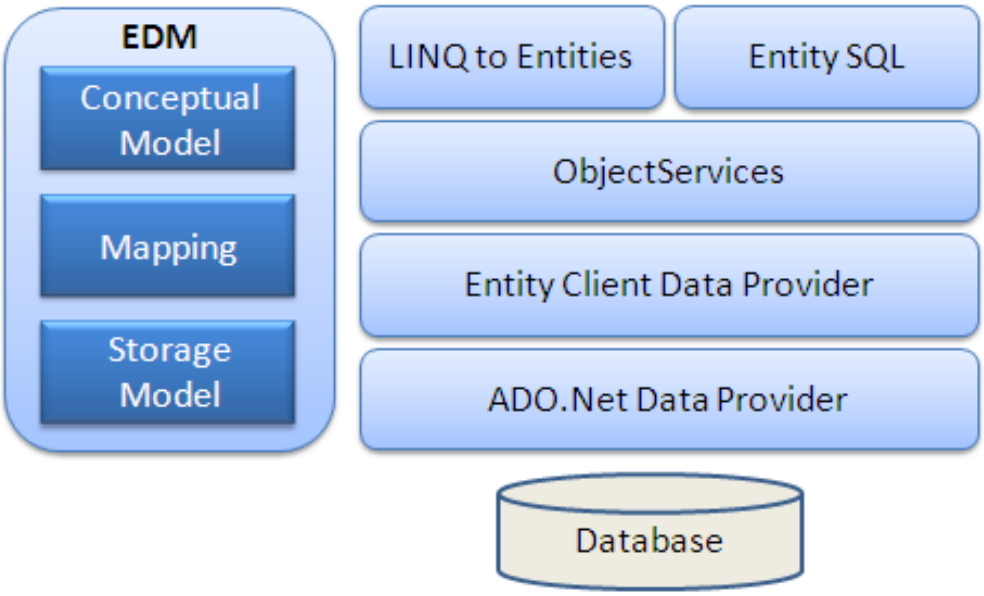


***Figure 1**. Entity Framework Architecture Source (EntityFrameworkTutorial.net, 2015)*

Conceptual Model: Model classes and relationships
Storage Model: Database model: tables, views, relationships, keys
Mapping: Translation of Conceptual Model into Storage Model
LINQ to Entities: Query language of treating entities as Objects
Entity SQL: Query language

Object Service: Translation of data from database into objects and vice-versa. Responsible for data access.
Entity Client Data Provider: Translation of LINQ to Entities and Entity SQL to plain SQL.
ADO.Net Data Provider: Communication with the database using ADO.Net. (EntityFrameworkTutorial.net, 2015)

The EDM file can be generated through
• The EDMGen command-line tool
• ADO.NET data model wizard in Visual Studio
• Creating the model in Visual Studio and then generate your database structure from this model. (Mackey, 2010)

The phases of creating the model by GUI are self-explanatory, as the windows guide in a simple way through all the process, by selecting the databases which we are modeling and then choosing the tables that we want to include.

1.3 Entity Framework Development Approaches

The most popular development approach is Database – First approach, which is based on generating object relational mappings according to an existing database. Therefore all the rules, connections, triggers, relationships, methods are defined in the database level, and we just call them to our Entity Framework and use them. Moreover we can add more capabilities to our model by creating classes and methods that are not part of the database. This is called POCO support (Plain Old CLR Objects). (Mackey, 2010)

A good work practice would be not to update the automatically generated database classes, so we will not have to rewrite the changes any time we generate database files. Another reason is that keeping them separately will be more helpful when we change our application and we need to update the functionalities. POCO feature is quite powerful because we can shape our existing database depending on our application requirements, without changing physically the rules of the database.

The other approach of data accessing is Model - First way. There is no database in this approach. The programmer creates the model first on Entity Framework using GUI provided by the platform. Relationships, keys, connections are defined by the programmer, but not the tables, rows, tuples. It is only the conceptual model needed, the translation into physical organization is done by Entity Framework. This option is quite interesting as it is creative and flexible. Abstraction is the strongest point of this option.

There exists also another approach which is Code – First approach. This way is more programming oriented, it does not use any GUI or given workflow. The programmer writes domain classes, which will be translated into tables in the actual model.

The following figure illustrates how the translation is performed, the first case is Database – Driven approach where tables are translated into classes, the second case is the Code – First Approach

where Domain Classes are translated into tables and the last case is dedicated to Model – First where we define everything in our DB model (using the provide GUI), so it is translated into a database and classes.

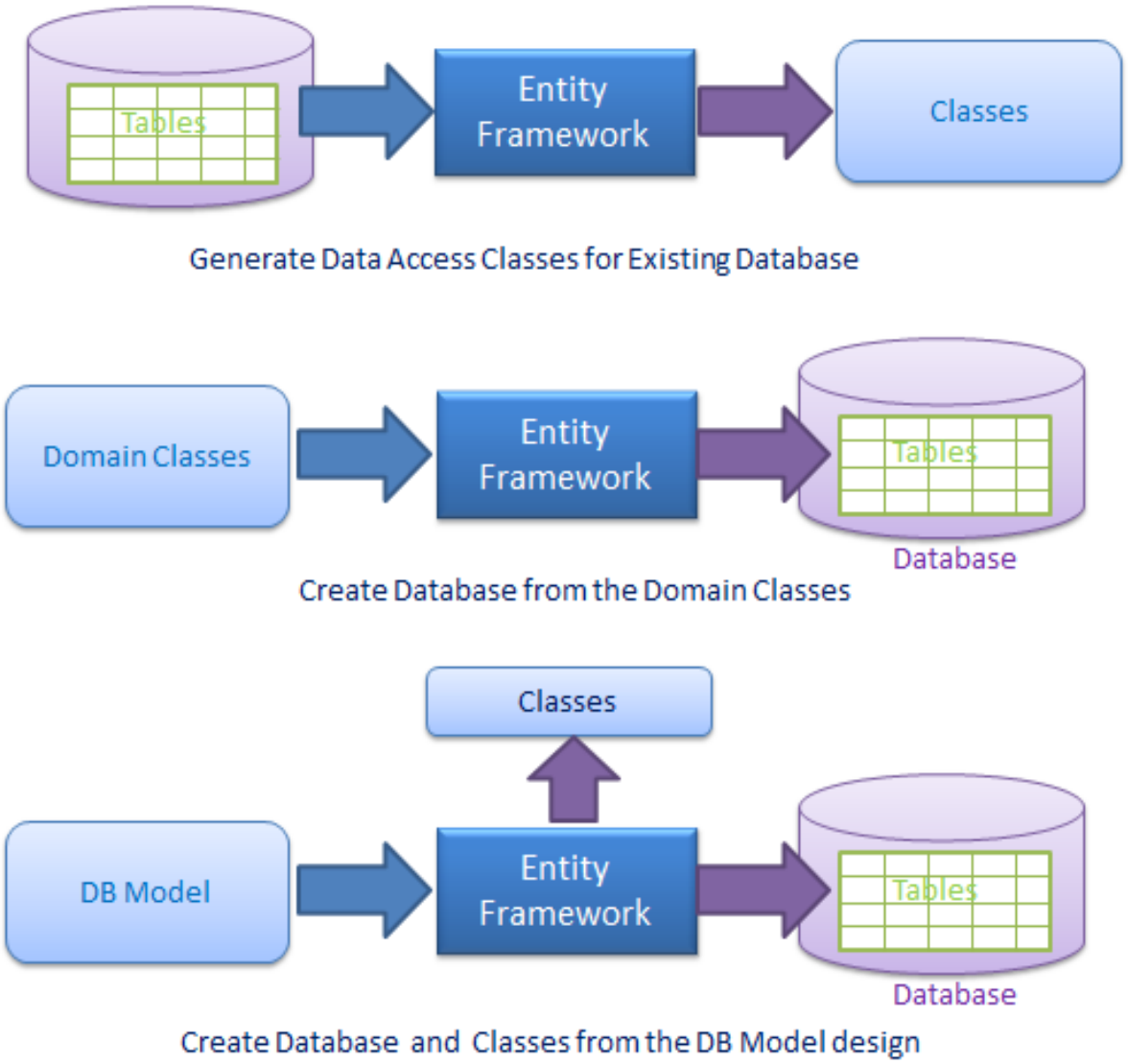


Figure 2. *Entity Framework cases, Source (EntityFrameworkTutorial.net, 2015)*

1.4 Entity Framework data loading

Entities in Entity Framework are more conceptual than physical. So while loading an entity, we want to access it as a whole, as a unit. If we were in traditional Relational Database Model, an entity could be divided among different tables. So if we need to access it, we will have to join on

the keys of all tables it is part of, in order to see the entity as a whole. This requires a good knowledge of the database model.

Entity Framework offer the possibility to access all the features of an entity easily, without the need of join queries. By typing the entity name then "." and then the attribute, the join between tables is done automatically. For example: customer.orders will return the orders of the entity customer, even though they might be in different tables.

There are two main ways of loading the data in Entity Framework. The first one is Eagerly Loading. This way uses the concept of an entity as a whole and so loads the entity entirely. This means that when we call a customer, the links are called and the loading of other attributes related to the customer is done automatically. However the main drawback of this way is that we might load unnecessary data every time we call an entity and we slow our process. Therefore the lazy loading way is offered too. Lazy loading does not load the attributes until you call them. So when we call customers, their relation with the orders is still there but is not loaded until we type customer.orders. Lazy loading is more efficient as it does not load the memory with unnecessary information. The difference is the performance is considerable. Lazy loading can be turned off or on by the user.

1.5 Entity States

Entities might have different states during their life. They can be inserted, updated, deleted and Entity Framework keeps track of the changes. We can modify an entity in our code, put until we run Save Changes, nothing is performed physically on the database. However we are keeping track of our entities, even though the actions are not performed. The changes of the state are kept within the context we are working on, if the context is updated, we start keeping track again and we lose the previous information. For example if we delete a customer, the state of this entity will be deleted. When we run Save Changes, then this state not show anymore, as the entity does not exist and we do not keep any track any more.

The states an entity can have are:

- Added (when we create a new entity)
- Deleted (when the entity is deleted)
- Modified (when the entity is updated)
- Unchanged (when we do not change anything in the entity)
- Detached (when we do not keep track any more regarding the changes that might happen with the entity, the entity is not attached any more)

The following schema provides a clear view of how we keep track of our entities:

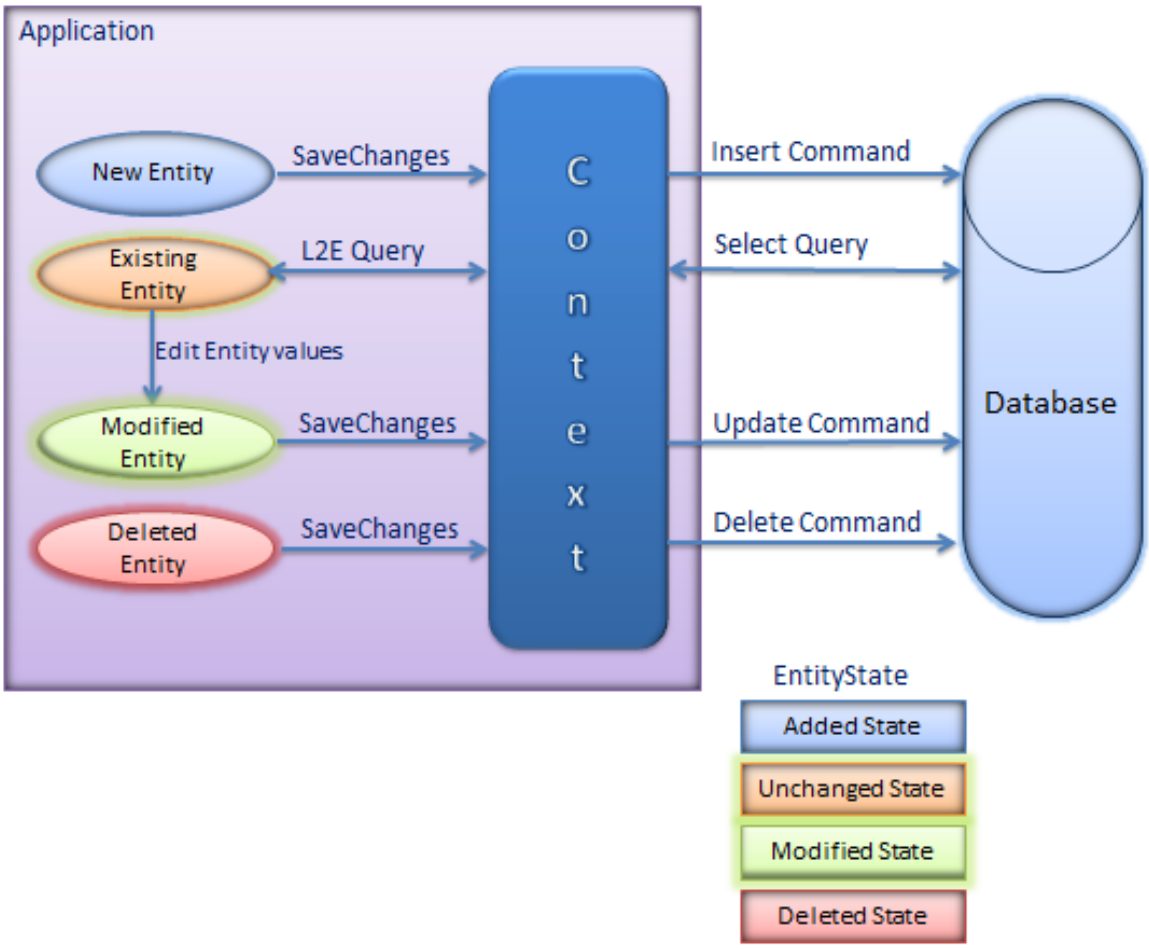


***Figure 3**.Entity States Source (EntityFrameworkTutorial.net, 2015)*

1.6 Queries execution

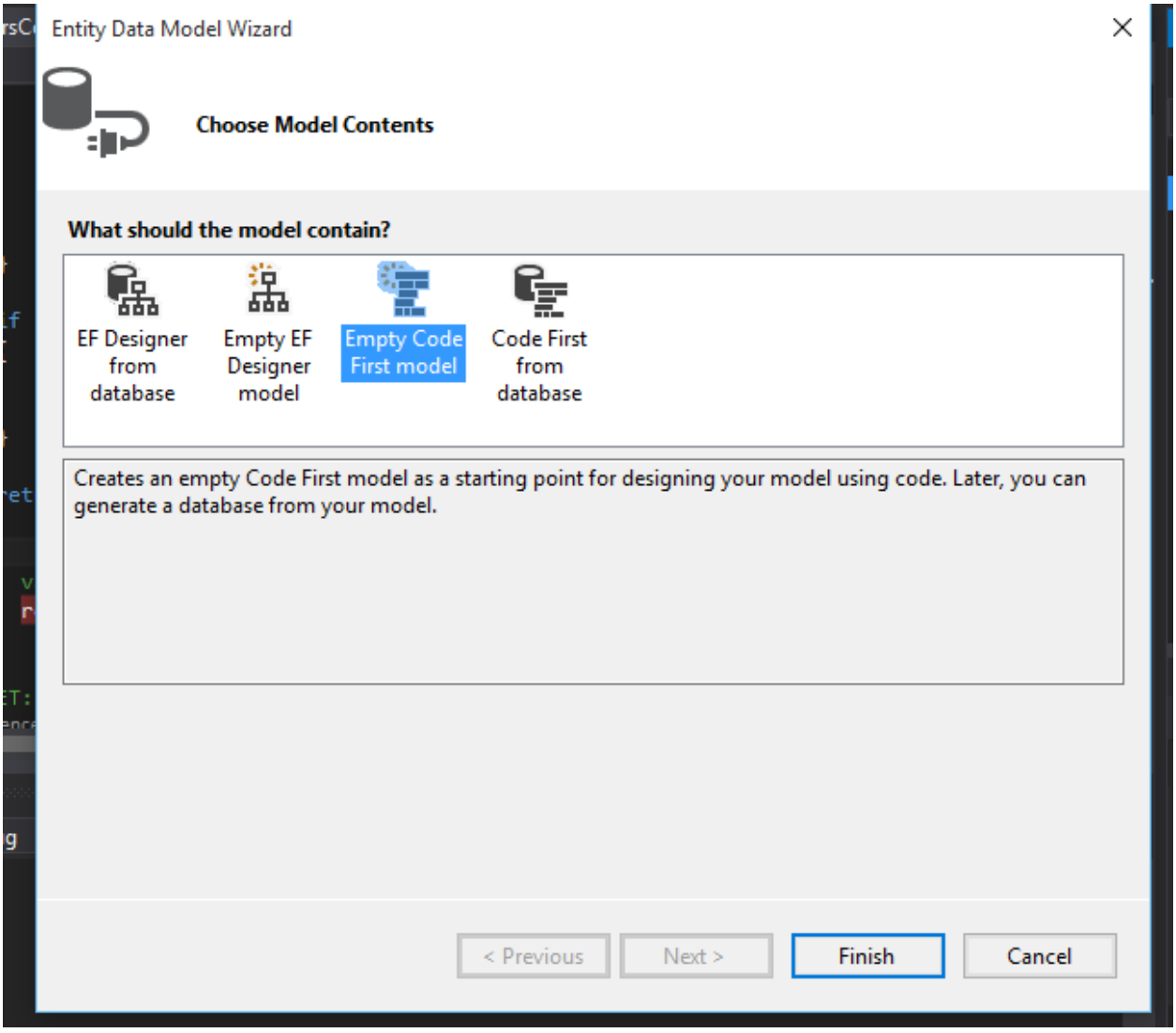
Queries in Entity Framework follow a set of operations. First of all, the metadata of the model is loaded and the connection to the database is established. These processes are considered of a moderate cost. Afterwards, a set of local views are generated and this phase has a high cost. Then the preparation of query (pre compile) and the execution of it are performed, with moderate and low cost respectively. The last phase of loading and validating the types, tracking (keeping record of the changes we make in our model) and materializing the objects has a low cost. Therefore it is quite obvious that the generation of the local views is the most costly action in this process it is highly recommended to pre-generate them and add to the project. . (Microsoft, 2015)

There are two ways of query execution, the deferred and the immediate execution. In the deferred execution, no query is executed while written, only when the statement “for each” is called. Moreover, the queries do not store any result and we can use them several times, while the immediate execution performs the executing for any query that returns a value. (Klein, 2010)

2. Implementations in Entity Framework

2.1Code First

In the code first approach the programmer decides on the entities that he wants to have in his system and the relationships they have. In order to do this you have to add an ADO.Net entity data model into your project. Then select empty code first type.



You can see the created model class which is extended from the DbContext Super Class. You can define the Entities that you have in your project as DbSet Collections.

```
public class Model1 : DbContext
   {
       public Model1() : base("name=Model1") { }
```

```
        public DbSet<Customer> Custoemrs { get; set; }
        public DbSet<CustomerDemography> CustomerDemographies { get; set; }
        public DbSet<Order> Orders { get; set; }
        public DbSet<Product> Products { get; set; }
        public DbSet<ProductCatagory> ProductCategories { get; set; }

    }
```

The most essential part of any relational database is the relationships. You can configure the different types of relationships between the entities as follows. All the connections are depicted as virtual properties of the class.

2.2 One to many

In the following scenario a Product belongs to a certain category.

```
public class Product
    {
        public int ProductId { get; set; }
        public string ProductName { get; set; }
       [ForeignKey("Catagory")]
        public int CatagoryId { get; set; }
        public virtual ProductCatagory Catagory { get; set; }
    }
```

There are many products that belong to a certain category which is represented as a list.

```
    public class ProductCatagory
    {
        public int ProductCatagoryId { get; set; }
        public string CatagoryName { get; set; }
        public virtual List<Product> Products { get; set; }
    }
```

2.3 Many to Many

In a many to many relationships both classes have virtual lists of the other class. But on the weak relationship you need to have the identity column defined.

```
public class Customer
    {
        public int CustomerId { get; set; }
        public string Name { get; set; }
        public virtual List<CustomerDemography> Demographies { get; set; }

    }
```

```
 public class CustomerDemography
{
    [Key]
    public int DemographyId { get; set; }
    public string Name { get; set; }
    public virtual List<Customer> Customers { get; set; }
}
```

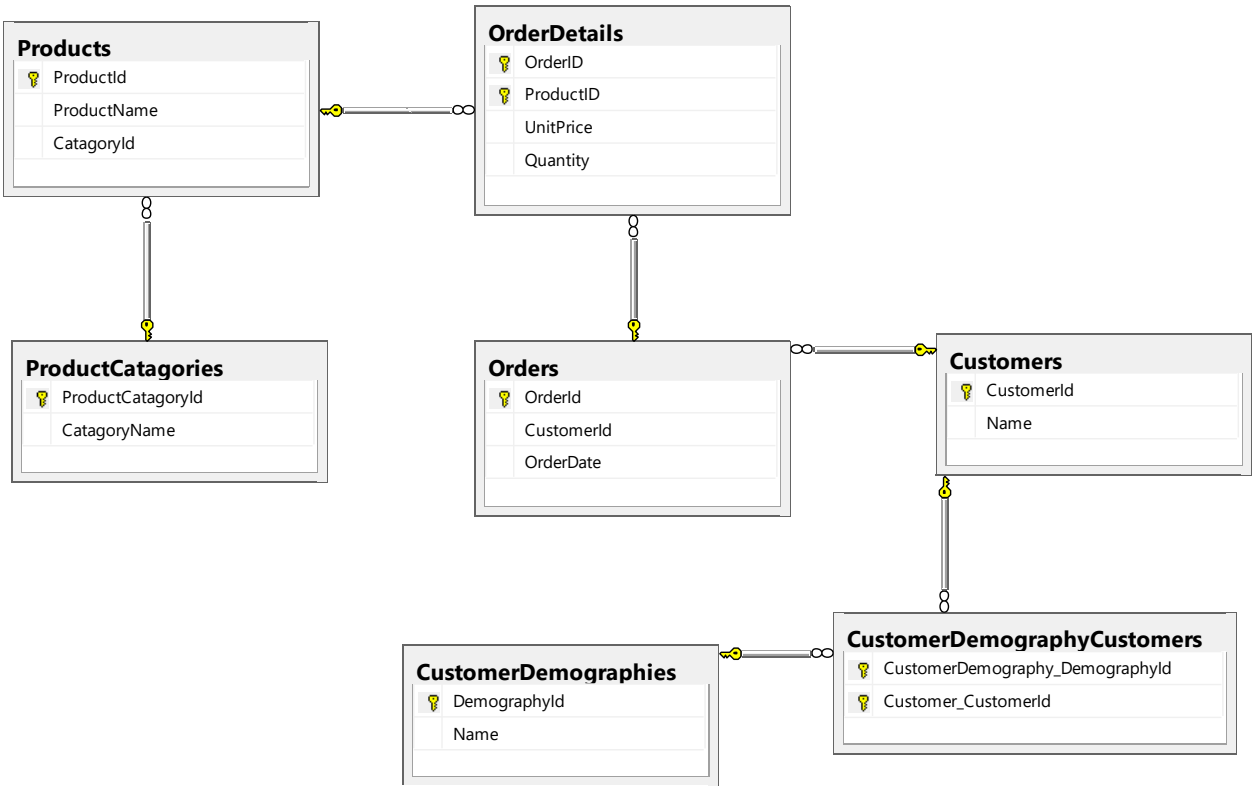
2.4 Many to many with additional attributes

If there is a many to many relationship with additional attributes (Product and order with details), you need to create the connecting table as an entity. And have one to many relationships to it from the main tables.

```
public class Product
{
    public int ProductId { get; set; }
    public string ProductName { get; set; }
    public List<OrderDetails> OrderDetails { get; set; }
}


public class Order
{
    public int OrderId { get; set; }
    public int CustomerId { get; set; }
    public DateTime OrderDate { get; set; }
    public List<OrderDetails> OrderDetails { get; set; }

}

public class OrderDetails
{
    [Key, Column(Order = 0), ForeignKey("Order")]
    public int OrderID { get; set; }
    [Key, Column(Order = 1), ForeignKey("Product")]
    public int ProductID { get; set; }
    public decimal UnitPrice { get; set; }
    public int Quantity { get; set; }
    public virtual Order Order { get; set; }
    public virtual Product Product { get; set; }

}
```

For the connecting table there are virtual attributes for the main tables. In order to maintain the proper relationships and the composite primary key the data attributes Key and foreign key is used giving the reference to the virtual object name.

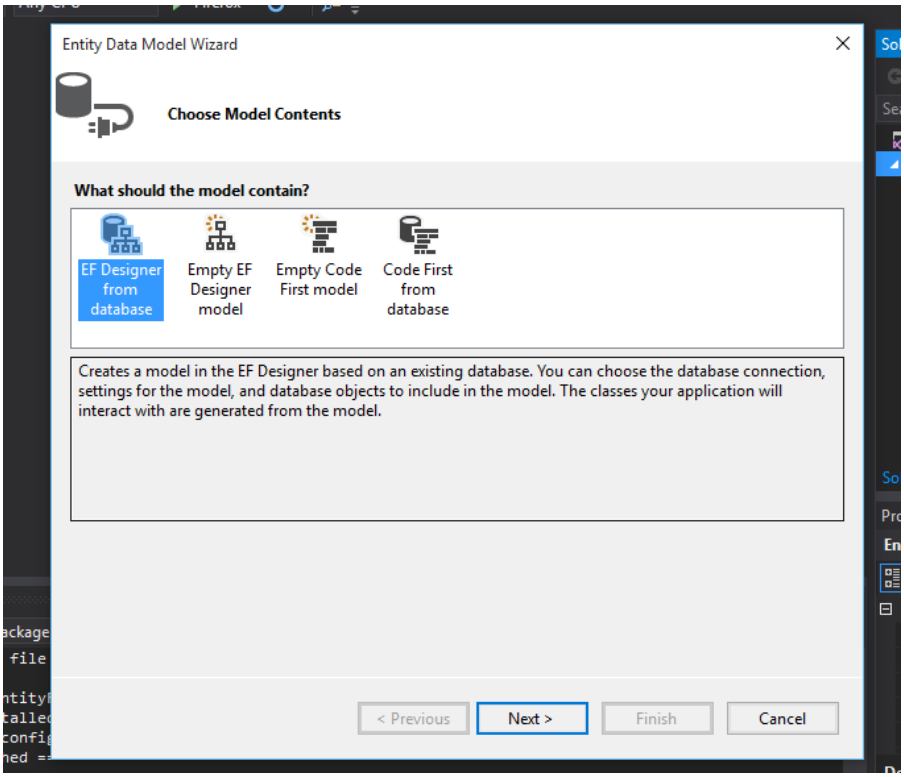
Once you have the code setup the entity framework will generate the tables when you run the code for the first time. The structure created for the above code first relationships is as below.

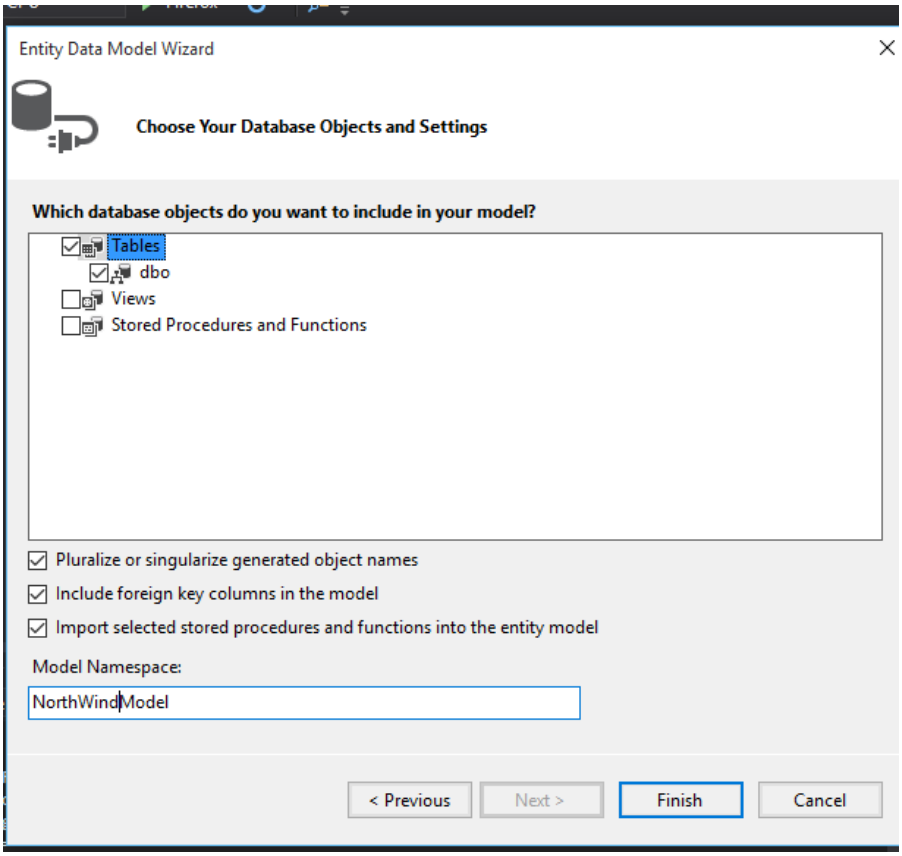


2.5 DB First

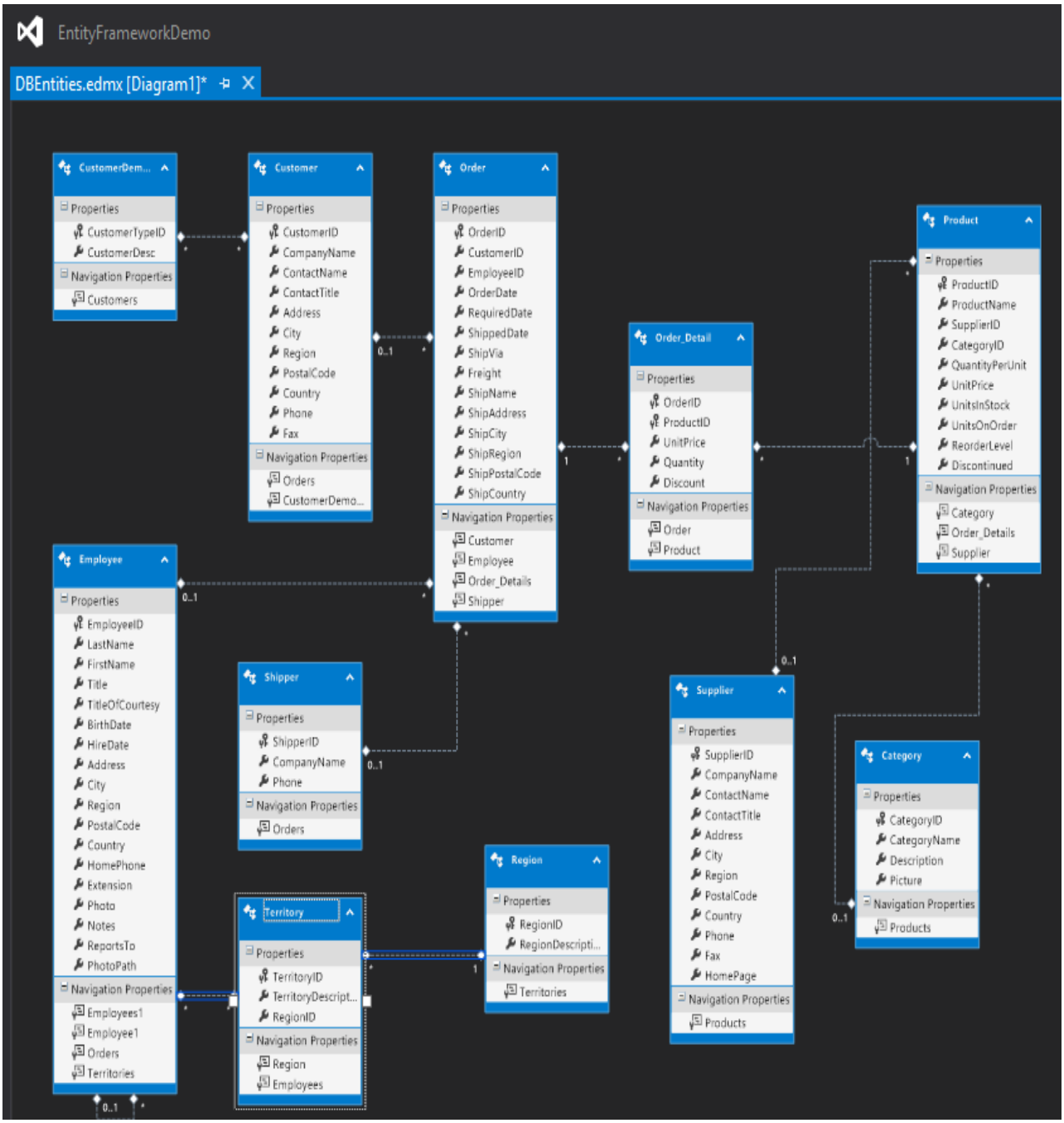
Most of the time the design of the table structure is decided before the start of the actual project or may be someone want to create a project for an existing database. In this scenario you can use the DB first approach in entity framework. This is more like the natural approach which current projects use and have more flexibility with the GUI interface

You need to add an ADO.Net entity data model for this with EF designer from database for this. Then you will be prompted to connect to the database you want to connect. After that all the tables, views and stored procedures in the database will be presented and you can pick the ones that you want to work with.

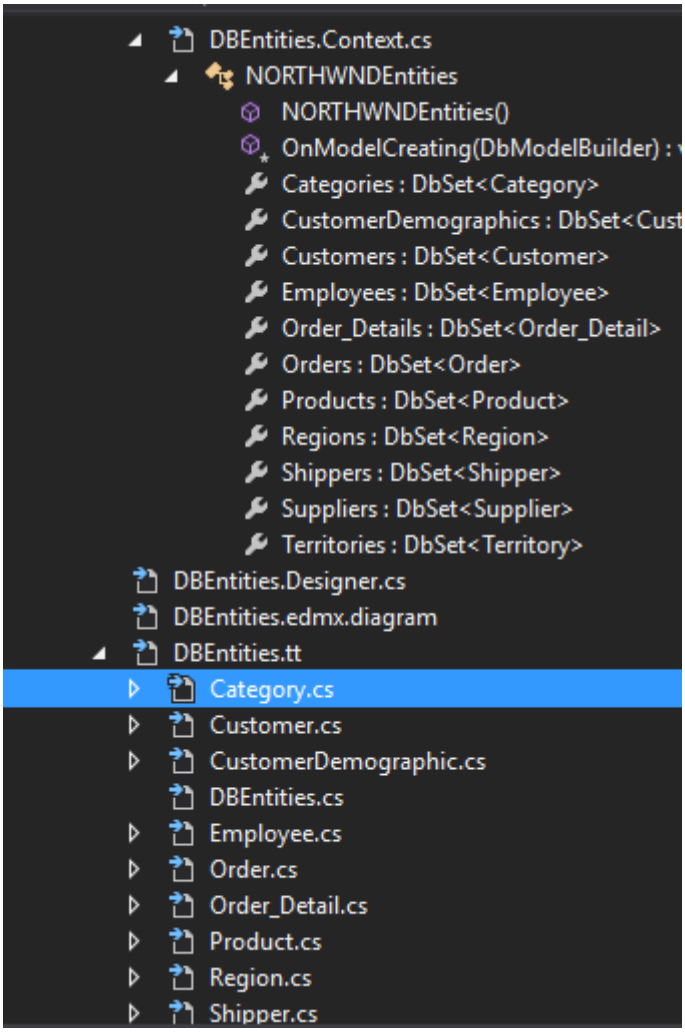

Entity Data Model Wizard
Choose Model Contents
What should the model contain?
EF Designer from database
Empty EF Designer model
Empty Code First model
Code First from database
Creates a model in the EF Designer based on an existing database. You can choose the database connection, settings for the model, and database objects to include in the model. The classes your application will interact with are generated from the model.
< Previous
Next >
Finish
Cancel


Entity Data Model Wizard
Choose Your Database Objects and Settings
Which database objects do you want to include in your model?
Tables
dbo
Views
Stored Procedures and Functions
Pluralize or singularize generated object names
Include foreign key columns in the model
Import selected stored procedures and functions into the entity model
Model Namespace:
NorthWindModel
< Previous
Next >
Finish
Cancel

For this example the Northwind database was chosen (https://northwinddatabase.codeplex.com/) and the following designer and all the classes were created for the schema.



Through the designer you can see all the relationships that are there in the schema and the cardinalities.



2.6 Queries with entities

With entity framework you can query the entities mainly using two LINQ syntaxes. The method syntax or the query syntax. In this report the main syntax used is the method syntax.

1. Selection

- Get All Employees

With LINQ Method syntax

```
private NORTHWNDEntities db = new NORTHWNDEntities();

            var employees = db.Employees;
            return View(employees.ToList());
```

LINQ Query syntax

```
var employees = from e in db.Employees select e;
```

- Get Employee with the name “Smith”

```
db.Employees.Where(e=>e.FirstName=="Smith");
```

2. Projection

With Entity Framework you can project the result into custom objects

```
            db.Employees.Where(e => e.FirstName == "Smith")
.Select(e=>new {name=e.FirstName, age=DbFunctions.DiffYears(DateTime.Now,e.BirthDate)
});
```

The Dbfunctions class contains EDM canonical functions for use in DbContext or ObjectContext LINQ to Entities queries.

3. Joining

By default Entity Framework uses lazy loading to load the entities. Therefore if you want to have the connections between the entities they need to be included in the query time. After that you can navigate through the navigational properties of the entity class to go to the relational entities.

```
db.Orders.OrderBy(o => o.Customer.CompanyName).Include(o => o.Customer).Include(o =>
o.Employee).Include(o => o.Shipper)
```

4. Adding

When adding an entity to your database you first have to create it and then add it to the corresponding DBSet. But it will not be added to the database until the save changes is called through the entity framework. If you want to add relationships you can add the parent to its DBSet and get the primary key of the parent (It will be automatically updated by the entity framework when you save the changes) and put it as the foreign key for the child relationship. Or you can add the parent entity to the child’s collection and then add the child to the corresponding DBSet

```
      Supplier sup = new Supplier
           {
               CompanyName = "New Supplier",
               City = "Brussels",
               Country = "Belgium",
           };

           db.Suppliers.Add(sup);
           Category cat = new Category
           {
               CategoryName = "New Category",
```

```
    Description = "This is a new Category"
};

Product newProduct = new Product
{
    Category = cat,
    ProductName = "New Product",
    ReorderLevel = 10,
    SupplierID=sup.SupplierID
};

db.Products.Add(newProduct);
```

In the above example a new product is added with a new supplier and a new category. The category is added with the product and the supplier is added first and then its Id is taken and inserted in to the product.

5. Looking up with the primary key and Editing

If you want to edit an entity you can take it from the database and do the necessary manipulation on the entity and then call save changes to write the changes to the database. If you have a primary key of an entity you can use the "Find" method of the DBSet to find the entity by the primary key.

```
Product p = db.Products.Find(1);
            p.UnitPrice=(decimal)150.99;
            db.SaveChanges();
```

6. Deleting

When deleting an entity you have to remove it from the DBSet collection and then save the changes. If there are any constraints that are there in the schema it will throw an error when you try to save the changes.

```
Order order = db.Orders.Find(2);
            db.Orders.Remove(order);
            db.SaveChanges();
```

7. Turning off tracking

As explained in the details the entity framework keeps track of all the entities that are retrieved and being used by the program. Sometimes this could be an unnecessary overhead depending on the scenario. If you have a list of entities that you need to display in a static page, read-only there is no point of keeping track of the changes done to them. For this when you can remove the tracking from the entities when you retrieve them. When you do this none of the changes that you do will reach the database even if you save the changes as it is not being tracked.

```
orders = db.Orders.Take(100).Include(o => o.Customer).Include(o => o.Employee)
             .Include(o => o.Shipper).AsNoTracking();
```

8. Query in the DB vs Query in the Program

With entity framework it is possible to do some of the filtering of the data in memory of the application (not the DB server) through the IEnumerable interface. In this the data is loaded to the memory and filtered in the memeory. So under heavy load this could slow down the performance considerably. If you use the Iqueriable interface the entity framework will build a query and send it to the database for the filtering. The following examples show how different techniques send different queries to the database.

```
IQueryable<Employee> employees = db.Employees;
 employees = employees.Where(e => e.FirstName == "John");
```

The resulting query sent to the database

```
SELECT
    [Extent1].[EmployeeID] AS [EmployeeID],
    [Extent1].[LastName] AS [LastName],
    [Extent1].[FirstName] AS [FirstName],
    [Extent1].[Title] AS [Title],
    [Extent1].[TitleOfCourtesy] AS [TitleOfCourtesy],
    [Extent1].[BirthDate] AS [BirthDate],
    [Extent1].[HireDate] AS [HireDate],
    [Extent1].[Address] AS [Address],
    [Extent1].[City] AS [City],
    [Extent1].[Region] AS [Region],
    [Extent1].[PostalCode] AS [PostalCode],
    [Extent1].[Country] AS [Country],
    [Extent1].[HomePhone] AS [HomePhone],
    [Extent1].[Extension] AS [Extension],
    [Extent1].[Photo] AS [Photo],
    [Extent1].[Notes] AS [Notes],
    [Extent1].[ReportsTo] AS [ReportsTo],
    [Extent1].[PhotoPath] AS [PhotoPath]
    FROM [dbo].[Employees] AS [Extent1]
    WHERE N'John' = [Extent1].[FirstName]
```

With IEnumerable

```
       IEnumerable<Employee> employee = db.Employees;
        employee = employee.Where(e => e.FirstName == "John");
```

The Query is

```
SELECT
    [Extent1].[EmployeeID] AS [EmployeeID],
    [Extent1].[LastName] AS [LastName],
    [Extent1].[FirstName] AS [FirstName],
    [Extent1].[Title] AS [Title],
    [Extent1].[TitleOfCourtesy] AS [TitleOfCourtesy],
    [Extent1].[BirthDate] AS [BirthDate],
    [Extent1].[HireDate] AS [HireDate],
```

```
    [Extent1].[Address] AS [Address],
    [Extent1].[City] AS [City],
    [Extent1].[Region] AS [Region],
    [Extent1].[PostalCode] AS [PostalCode],
    [Extent1].[Country] AS [Country],
    [Extent1].[HomePhone] AS [HomePhone],
    [Extent1].[Extension] AS [Extension],
    [Extent1].[Photo] AS [Photo],
    [Extent1].[Notes] AS [Notes],
    [Extent1].[ReportsTo] AS [ReportsTo],
    [Extent1].[PhotoPath] AS [PhotoPath]
    FROM [dbo].[Employees] AS [Extent1]
```

By comparing the two queries it is clear that when you use the Queryable interface the querying is done in the database and Enumerable does it in the application.

9. Extension methods

The extension methods in Entity Framework can be compared to having views in the database. These methods can be used to retrieve queries which you use frequently rather than writing them over and over again. (Getting non discontinued products)

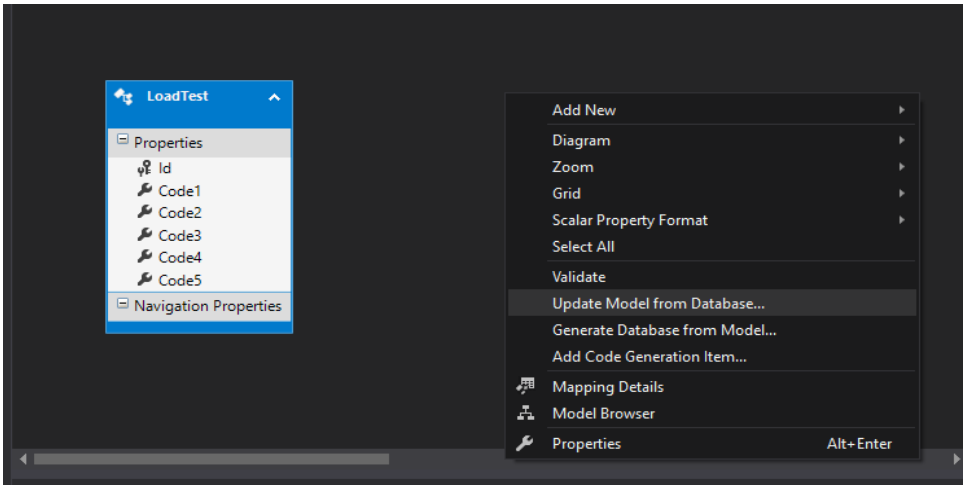
```
public static class Extensions
 {
 public static IQueryable<Product> NonDiscontinued(this IQueryable<Product> products)
             {
                    return products.Where(p=>!p.Discontinued);
             }
  }
```

You need to write the extension methods as static in a static class. Then use it in the query as follows.

```
db.Products.NonDiscontinued();
```

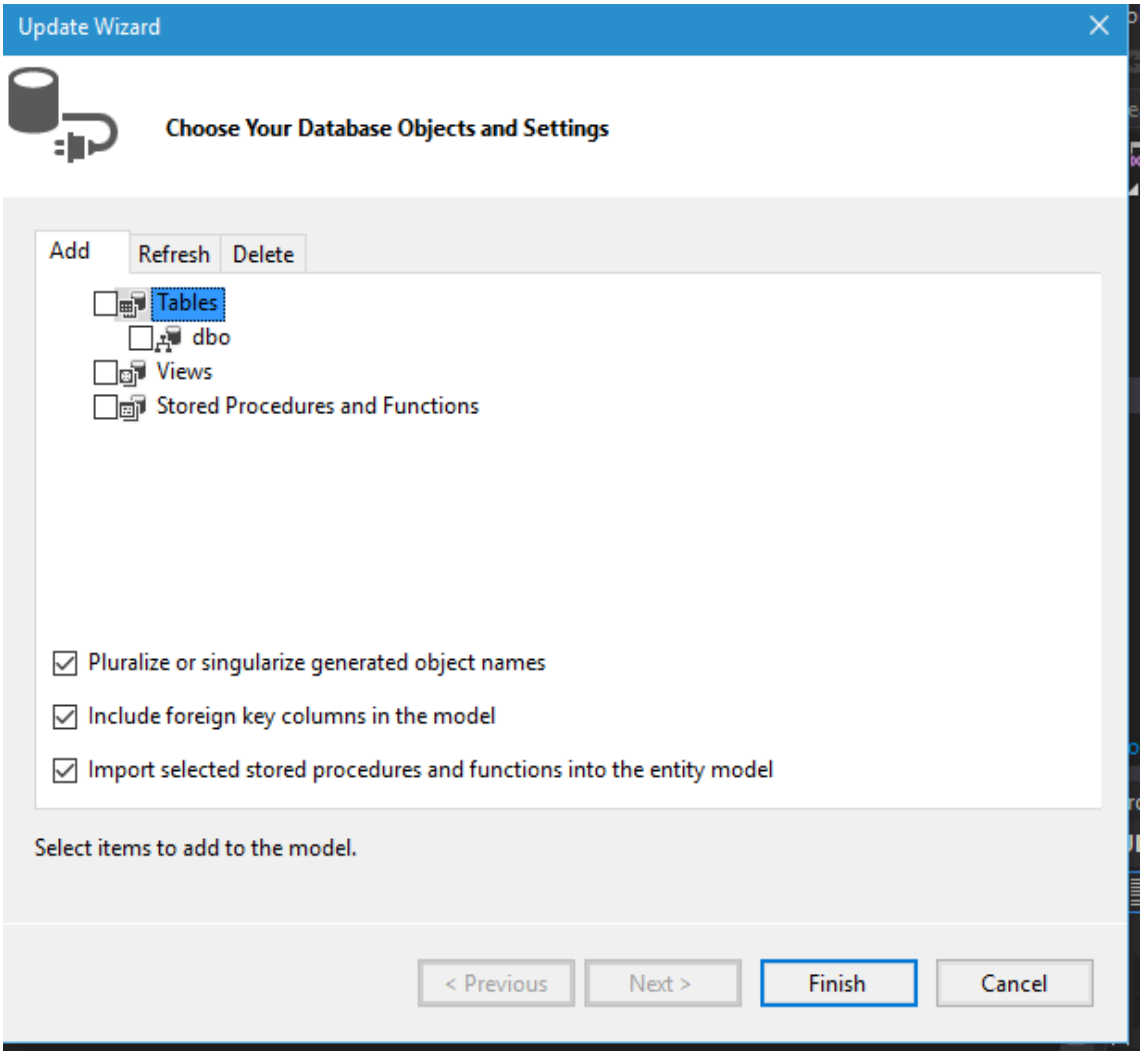
10. Migrating changes on the DB

If the changes are done in the DB you need to migrate the changes to your model.



Right click on the canvas of the edmx and select the option to update the model from the database.

Then you reach the following screen where it shows the added, deleted and changed elements in the database which you can select to include in your model.



- Updating the DB from the model with migration scripts

In the code first approach you can enable migration to your model and do the DB changes through the migration scripts (https://msdn.microsoft.com/en-us/data/jj591621). Since we used the DB first approach it is not possible to implement this.

3. Experiment: Performance Analysis

One of the known drawback of using Entity Framework is that the performance is remarkably low in case of large data. Therefore we will test our infrastructure in different scenarios and analyze it later.

We will enter data to the database from the Entity Framework, we will start with small data and then we will increase it step by step. In the meantime we will measure the time it takes to perform this operation (granularity milliseconds).

This is the code we are using to enter the data:

```
var db = new ULBEntities();
      for (int i = 0; i < 10; i++)
      {
          LoadTest lt = new LoadTest
          {
              Code1 = Guid.NewGuid(),
              Code2 = Guid.NewGuid(),
              Code3 = Guid.NewGuid(),
              Code4 = Guid.NewGuid(),
              Code5 = Guid.NewGuid()
          };
          db.LoadTests.Add(lt);
      }
      db.SaveChanges();
```

In the yellow part, we input different figures, from 10 to 50000.

The results are as follows:

Records	Milliseconds
10	1550
100	2210
1000	8650
5000	42960
10000	92960
20000	288500
30000	551260
40000	885020
50000	1163310

It is clear that when the data load increases the execution takes a considerably long time to complete. Considering 20 minutes to insert 50 000 rows is not an acceptable performance. Therefore while investigating on the issue we found out several factors affecting performance for bulk data entry

- The growing context

Each and every item that goes through the EF is added to the context. Therefore when we add new object the context expands which consumes a lot of memory.

- Tracking of objects

EF needs to keep track of every object in its context and it's listening to any changes happening to them.

- Bulk saving

It is better to save in batches rather than saving large bulk of records.

After the findings we optimized the code for the following one.

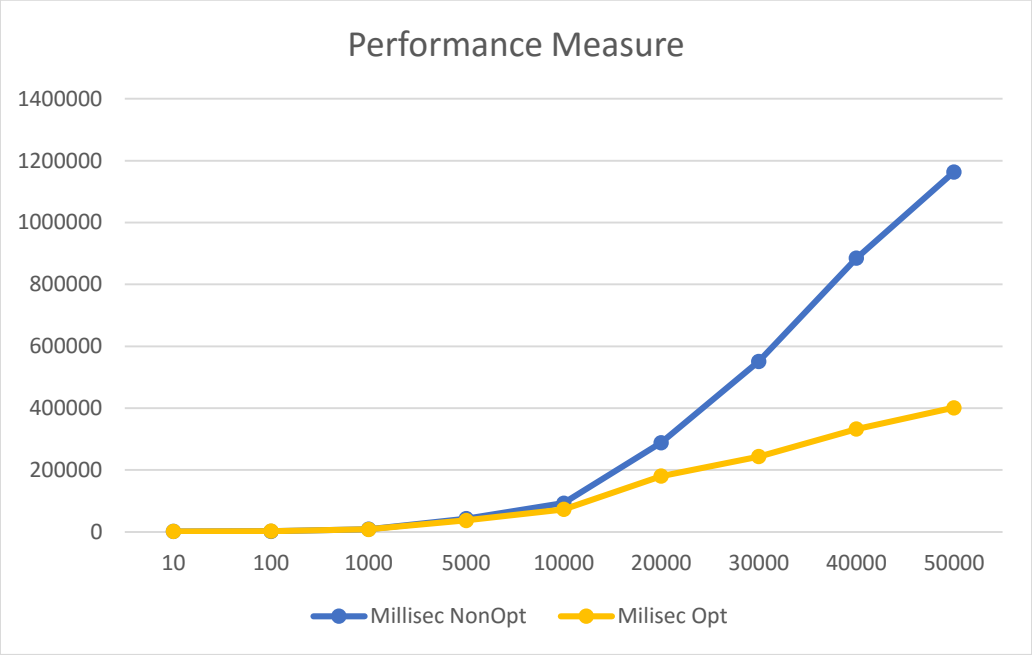
```
var db = new ULBEntities();
 db.Configuration.AutoDetectChangesEnabled = false;
 db.Configuration.ValidateOnSaveEnabled = false;
 for (int i = 0; i < 40000; i++)
 {
     LoadTest lt = new LoadTest
     {
         Code1 = Guid.NewGuid(),
         Code2 = Guid.NewGuid(),
         Code3 = Guid.NewGuid(),
         Code4 = Guid.NewGuid(),
         Code5 = Guid.NewGuid()
     };

     db.LoadTests.Add(lt);

     if (i % 1000 == 0)
     {
         db.SaveChanges();
         db.Dispose();
         db = new ULBEntities();
         db.Configuration.AutoDetectChangesEnabled = false;
         db.Configuration.ValidateOnSaveEnabled = false;
     }
 }
 db.SaveChanges();
```

We turned off the parameters to auto detecting changes and the validations before saving in the context. Furthermore we are saving in batches every 1000 records and after we save we dispose the context and make a new context since we don't need to keep the entered data.

The following is the result of the optimized code runtime.

Records	Millisec Opt
10	1630
100	2460
1000	8570
5000	37350
10000	73260
20000	180430
30000	243690
40000	332770
50000	401630

It is obvious that the improvement is significant. The graphical representation reflects the improvement better.



We can see that the behavior of both cases is similar of small data, but the difference is considerable for large data. We have tried to find a regression model for both cases, in order to find the coefficients of variables, a possible relation between them, but also to have the slope and intercept to predict future growth.

Therefore for the not optimized case, we could find the below coefficients for the intercept and slope, with an excellent P value of the slope, while for the intercept it is relatively acceptable.

	Coefficients	*P-value*
Intercept	-58739.6265	0.140112598
Slope	22.8369524	0.00000089392539

However the entire model has a good significance for the confidence level 95%:

ANOVA	
	Significance F
Regression	0.00000089392539

Regression Statistics	
Multiple R	0.986647971
R Square	0.973474219

For the optimized case we found a good entire model, a good coefficient of slope and for intercept still acceptable:

	Coefficients	*P-value*
Intercept	820.1494	0.834398731
Slope	8.16353	0.000000000208

ANOVA	
	Significance F
Regression	0.0000000002

Regression Statistics	
Multiple R	0.99878
R Square	0.997562

After both analysis, we considered it as interesting to divide the behavior of the regression in two different regression models. As we can see from the graph, after the value of 10.000 records, the behavior of the performance is changing dramatically and maybe a single equation of the whole is not quite fitting. So, we consider 10.000 records as a starting point for a new regression for both cases. The previous behavior is similar for both cases and will be removed from the observations, we will only use the data from 10.000 records and on for the next analysis.

More in detail, for the not optimized case, the model is better than the previous one, all figures are significant:

Regression Statistics	
Multiple R	0.99618946
R Square	0.99239344

ANOVA

	Significance F
Regression	0.000282205

	Coefficients	*P-value*
Intercept	-224956	0.0162465
Slope	27.3722	0.000282205

We can come to the conclusion that our hypothesis holds, it is a complete different behavior when it comes to large data.

And for the optimized case, the results are almost the same as the previous models, which means that the behavior of the small data and the large data can be included in one equation, as they are comparable, the growth has almost the same coefficient. This is an important finding, as it means that the optimization techniques that we used have been quite productive in reducing the time of the processing and making it relatively comparable to small data.

Regression Statistics	
Multiple R	0.99878
R Square	0.997562

ANOVA

	Significance F
Regression	0.0000000002

	Coefficients	*P-value*
Intercept	820.1494	0.834398731
Slope	8.16353	0.000000000208

To conclude, in order to predict the performance for big data, we can put the number of record in the place of x and evaluate y which is the time in milliseconds.

For not optimized case: **`y = 27.3722 * x - 224956`**

For the optimized case: **`y = 8.16353 * x + 820.1494`**

The slope of the not optimized is almost 3-4 times bigger than the optimized. Of course the intercept plays an important role too, but in case of very large data, the intercept will not have a significant effect and the difference will be driven more by the effect of the slope.

4. Entity Framework Pros, Cons and Usage

We have already discovered most of the topics related to Entity Framework and now it is the time to reflect and compare with other options, find the stronger and weaker points and evaluate the cases where Entity Framework would be a satisfying solution.

Pros of using Entity Framework:

- Joins between tables are made naturally, without explicit statements.
- Quicker development time
- No need to have a knowledge on the database tables and relations
- Manipulation of objects is simple, no need for long query-s of update or delete.
- Entity Framework can work with different database providers (SQL Server, Oracle etc)
- Models and files can be shared within users and executed quite easily.
- Adaptive to different application development, you can create new methods and functions related to the requirements without changing the database itself or adding new stored procedures.
- Easy syntax, comparable to SQL but obviously simpler than object – oriented databases, in which knowing the syntax is a strong requirement.

In particular, advantages over LINQ to SQL:

- Entity Framework, as previously mentioned, works with different database environments (SQL Server, Oracle, DB2), while LINQ to SQL can work only with SQL Server.
- Entity Framework can support complex types, while LINQ to SQL cannot.
- Entity Framework can create a database from a model (Model-First approach, explained in paragraph 1.3) while LINQ to SQL cannot.
- Entity Framework has the capability of creating different types of relationships between its classes and relational tables (one to one, one to many and many to many) while LINQ to SQL can allow only one to one mapping. (Chauhan, 2014)

Advantages over NHibernate:

- In NHibernate there is no pure way of Code-First Approach, the actual way includes the involvement of third parties.
- Entity Framework supports schema migration, while NHibernate can only support the initial schema.
- NHibernate does not provide Asynchronous Operations, while Entity Framework does.

- NHibernate offers the flexibility in the languages it supports, but this is not considered an advantage, since some of them are not matured and they do not function as expected. Therefore Entity Framework is more stable.
- Documentation related Entity Framework is updated and covers almost all the issues, while in this point NHibernate's documentation is quite poor. (Kucinskas, 2014)
- Both of them offer the opportunity to track changes of their entities or not, but Entity Framework has an extra option: self-tracking entities. These entities decide to track changes on them only when they are connected to the context, so they switch between tracking and non-tracking smartly by themselves.
- Entity Framework offers more integration with Visual Studio, ASP.NET, WCF libraries. (Allen, 2012)

Advantages over ADO.NET:

- In Entity Framework the code is less than in ADO.NET, as for accessing the data, the model is uses, not the classes particularly. Therefore the code is more maintainable too.
- Operations with entities are easier and faster (update, delete, insert)
- Faster development time.
- Changes in the database are maintained directly by the model, no need to data access logic. (Patel, 2014)

We have almost a clear view what are the strong point of using Entity Framework. Let's discover now the dark side of the Entity Framework:

- It needs some time to disconnect from thinking in a traditional way of databases. Since SQL and relational databases are quite popular, switching to this new approach needs some spirit in the beginning.
- As mentioned in this report, Entity Framework is a layer on the database. Therefore it is apparent that more layers slow down the data exchange. Querying directly the database is not the same as writing a code in a language that will be translated to SQL to be executed to the database. This drawback cannot be distinguished for small data. But with a lot of data, the process really gets slow. According to the findings of the experiment in chapter 3 of this study, large data load is a disaster without performance optimization techniques. We would recommend not using Entity Framework for big data load at all.
- Every time there is a change on the existing database, the files need to be regenerated.
- Another remark during exploring Entity Framework is that given SQL functions cannot be called directly, functions as datediff etc. The way of accessing them is using the library DbFunctions then citing the name of the function. This is not a pure drawback, but those who are used to SQL, they might find it annoying.
- While working with entities, until we push our changes to the database, we keep the actions in the memory. This might cause a memory overload time after time, so we will have to run SaveChanges frequently.

Given the above outcomes, Entity Framework has a defined profile that should be used appropriately. Therefore, we would suggest it in developing environment, creating applications which need interaction with a database. Programming teams will adapt easily to the platform and they will find the abstraction offered quite helpful. No need for database deep knowledge. The connection to the database is fast and the relation to coding is close. So it will still feel like they are coding, even though their objects are tuples and tables.

Dynamic and mixed environments will be still appropriate, since Entity Framework offers the option to deal with different data sources and the integration capability is high.

We do not find it appropriate for business teams or even for retrieving or querying data, the usage of an Entity Framework is unnecessary, as its scope is wider than just daily interactions with database.

Moreover, we do not recommend it for large data processes as the performance degrades considerably.

To sum up, Entity Framework is a powerful tool, it can fit perfectly some gaps between database and programming environments, but it should not be used outside its scope.

References

Allen, J. (2012). *Comparing NHibernate and Entity Framework.* Retrieved from InfoQ: http://www.infoq.com/news/2012/06/NHibernate-EF

Chauhan, S. (2014). *Difference between LINQ to SQL and Entity Framework*. Retrieved from DotNet Tricks: http://www.dotnet-tricks.com/Tutorial/entityframework/1M5W300314-Difference-between-LINQ-to-SQL-and-Entity-Framework.html

EntityFrameworkTutorial.net. (2015). *Entity Framework Tutorial*. Retrieved from Entity Framework Tutorial: http://www.entityframeworktutorial.net/

Klein, S. (2010). *Pro Entity Framework 4.0.*

Kucinskas, D. (2014). *Entity Framework 6 vs NHibernate 4.* Retrieved from Devbridge Group: https://www.devbridge.com/articles/entity-framework-6-vs-nhibernate-4/

Mackey, A. (2010). *Introducing .NET 4.0 with Visual Studio 2010.*

Microsoft. (2015). *Performance Considerations (Entity Framework)*. Retrieved from Microsoft: https://msdn.microsoft.com/library/cc853327.aspx

Patel, B. (2014). *Advantage of Entity Framework over ADO.NET.* Retrieved from DotnetSpan.com: http://www.dotnetspan.com/2014/07/advantage-of-entity-framework.html