
LINQ & Entity Framework
Lesson 03 : EDM Tool Enhancement
Capgemini

Lesson Objectives

- In this lesson we will cover the following
  - Model First Development
  - Pluralization
  - Complex Types

Overview : Model First Development

- Model First Development allows us to build up a model from the scratch
- Model first Development allow a developer to create a new Model using the Entity Framework Designer.
- This will generate a database schema from the model
- The Model is stored in an EDMX File(.edmx extension) and can be viewed and edited in the Entity Framework Designer
- The Classes that you interact in the application are automatically generates form the EDMX Files

Model First Development allows to build up a Model from scratch . It allows you to create Entities, relationships, and inheritance hierarchies directly on the design surface of EDMX and then generate database from your model.

With the Entity Designer, we can define entities which are abstractions representing the objects in application domain. This is where we create a Entity Data Model (EDM) and is driven by an XML grammar from a model file extension of .EDMX

The Model First Approach was provided by Microsoft because it was one of the most often requested features by .Net Developers and System Designers. It is a more natural way to work compared to the **Database First** approach especially when designing the flow of data in the initial stages of a new project. Developers requested the flexibilty of creating the conceptual model first instead of going through the burden of outlining the project logic and then trying to design a database which would accomodate for all the data and information storage within the project. Using the **Model First** approach a developer can start working with the model of the databse and creating entities which make logical sense irrelevant of how they will actually be stored in the database in terms of tables.

From the model created, Visual Studio can then generate SQL statements referred to as *Data Defenition Language (DDL)* which the developer can use to execute on MS SQL in order to create the Database schema based on the designed model.

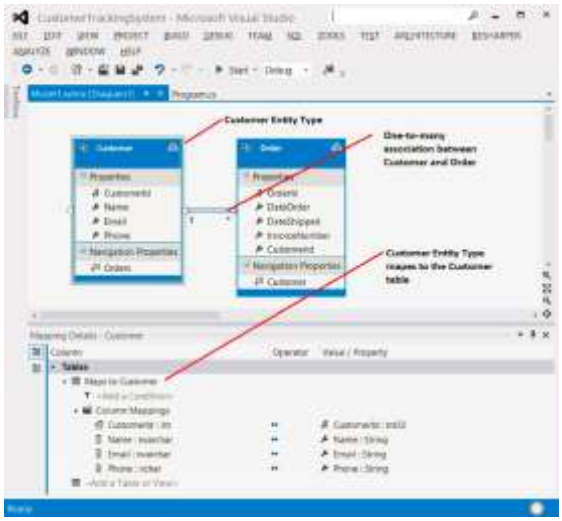
As an example, we might create an Order entity that will represent instances of orders and their details such as who placed the order, on what date and so on. Then we might create related entities to hold additional information such as the items in the order and their cost and quantities. The modeling experience gives a natural way to build up a description of the data in application visually, playing around with it and changing it easily The real power of this comes though when we want to turn the model into reality. That's because EF can take the EDM and generate two things from it, the Data Definition Language (DDL) to create a database as the model's concrete representation and data access code to manipulate it.

Overview : Model First Development(Contdd..)

From the model created, Visual Studio can then generate SQL statements referred to as *Data Defenition Language (DDL)* which the developer can use to execute on MS SQL in order to create the Database schema based on the designed model.

As an example, we might create an Order entity that will represent instances of orders and their details such as who placed the order, on what date and so on. Then we might create related entities to hold additional information such as the items in the order and their cost and quantities. The modeling experience gives a natural way to build up a description of the data in application visually, playing around with it and changing it easily The real power of this comes though when we want to turn the model into reality. That's because EF can take the EDM and generate two things from it, the Data Definition Language (DDL) to create a database as the model's concrete representation and data access code to manipulate it.

Model First Development (Contd..)



The above image shows the EDMX designer in Visual Studio .

The Image two entities in the designer i.e Customer and Order .

It show all the fields of the Entities and the relationship between them .

X.1: Model First Development

Model First Development

- Benefits of Model First Development
  - Visual Designer can be used to create a database scheme
  - Model can be easily updated as there is change in the database
  - No loss of data

Add the notes here.

X.2: Pluralization

Pluralization

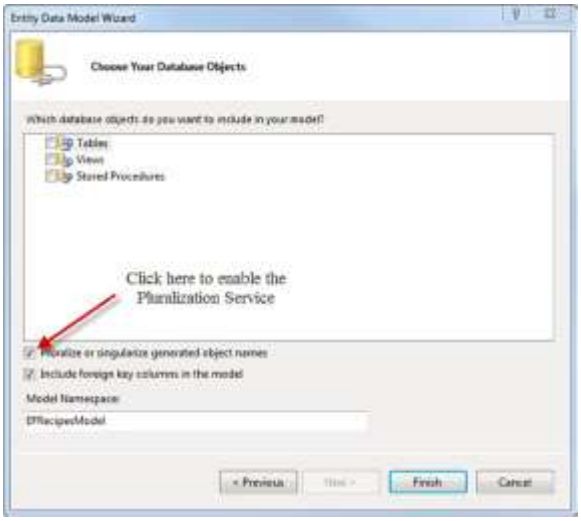
- Pluralization allows a developer to pluralize the name of Entities while create a Entity Data Model
- This allow the developer to differentiate between the Entity and the Entity Set
- This can be achieved while creating the model using the Model Designer by selecting a checkbox saying "Pluralize or Singularize generated object names"
- This can be also be done by code in Code First Approach

Pluralization :-

Pluralization allow a developer to Pluralize the name of Entity to represent an Entity Set.

X.2: Pluralization

Pluralization



The above screenshot show how to enable Pluralization in Model First and Database First approach

X.X: Model First Development

Demo

- Demo of Implementing Model First Development approach in Console Application.

Add the notes here.

X.3: Complex Types

Complex Types

- Complex types provide handy mechanism for storing and encapsulating properties related to one or more entities
  - Eg:- You may have more than one entity that needs to store phone and email information
- Complex types can also be used to add additional structure to your entities.
- They are made of scalar properties as well as additional complex types
- They can be instantiated from outside the parent entity and still provide the ability to navigate to them through the related entity and entities.

Complex Type:-

•Complex types are non-scalar properties of entity types that enable scalar properties to be organized within entities. Like entities, complex types consist of scalar properties or other complex type properties.
•Complex types allow you to group several properties into a single type for a property on an entity.
•A complex type can contain scalar properties or other complex types, but they cannot have navigation properties or entity collections.
•A complex type cannot be an entity key. Complex types are not tracked on their own in an object context.
•A property whose type is a complex type cannot be null. When you work with entities with complex type properties, you have to be mindful of this rule.
•Occasionally, when the value of a complex type property is unimportant for a particular operation, you may need to create a dummy value for the property so that it has some nonnull value.
•When you modify any field in complex type property, the property is marked as changed by Entity Framework, and an update statement will be generated that will update all of the fields of the complex type property.
•Complex type can also be with POCO (Plain Old CLR Objects).

X.3: Complex Types

Complex Types

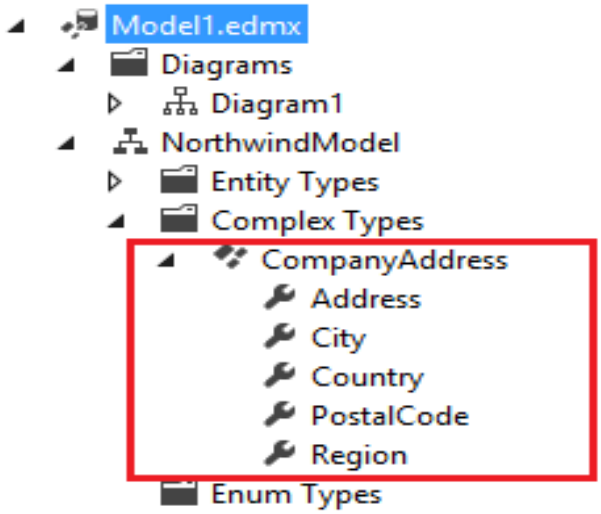
Complex Type:-

When you work with objects that represent complex types, be aware of the following:

•Complex types do not have keys and therefore cannot exist independently. Complex types can only exist as properties of entity types or other complex types.
•Complex types cannot participate in associations and cannot contain navigation properties.
•Complex type properties cannot be **null**. An **InvalidOperationException** occurs when **DbContext.SaveChanges** is called and a null complex object is encountered. Scalar properties of complex objects can be **null**.
•Complex types cannot inherit from other complex types.
•You must define the complex type as a **class**.
•EF detects changes to members on a complex type object when **DbContext.DetectChanges** is called. Entity Framework calls **DetectChanges** automatically when the following members are called: **DbSet.Find**, **DbSet.Local**, **DbSet.Remove**, **DbSet.Add**, **DbSet.Attach**, **DbContext.SaveChanges**,**DbContext.GetValidationErrors**, **DbContext.Entry**, **DbChangeTracker.Entries**.

X.3: Complex Types

Complex Types



The above screenshot shows a complex type CompanyAddress as a part of Model .

As you can see it is consisting of other types which make up CompanyAddress

X.X: Complex Types

Demo

- Demo for Creating and Using Complex Type in Entity Framework

Add the notes here.

X.X: [Topic]

Lab

- Lab Topic

Add the notes here.

Summary

- Summarize the topic in bullet points

Add the notes here.

Review Question

- Which of the following allows the development of model from scratch using model designer?
  - Design First
  - Code First
  - Model First
  - Database First

- Complex type cannot be used to add additional structure to the model
  - True
  - False

Add the notes here.

Review Question

- _____allow a developer to create a new Model using the Entity Framework Designer.
  - Code First Development
  - Designer First Development
  - Model First Development

Add the notes here.