Seanca 8

EF Database-First vs Code-First Approach vs Model-First Approach

- **Database-First approach**

Developer has to create the database first, model class will be auto-generated

- **Code-First Approach**

Developer has to create model class first, database will be auto-generated

- **Model-First Approach**

Developer has to design the model class first, database will be auto-generated

Database-First Approach

- Very useful if you have an existing DB or if you have DB designed by DBA separately
- Best for small-large projects, with fully known requirements
- You can make changes in ORM designer or partial classes
- You can change the database manually, and update the model classes from database
- Best, if you want to use stored procedures

Code-First Approach

- Very popular
- Best for small to large projects when requirements are not fully known(continuous changes)
- Full control over model classes, they are not auto-generated
- No need to worry for the DB, EF will handle DB creation
- Not good if you want to use stored procedures

Model-First Approach

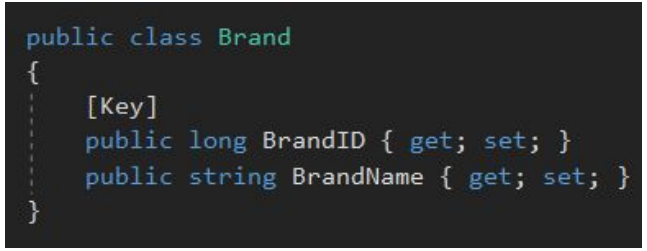
- Good for designer fans, not writing code or SQL
- The model is drawn using ORM designer in Visual Studio, and both the database and model classes will be auto-generated
- Suitable for small projects

Code-First Approach Example

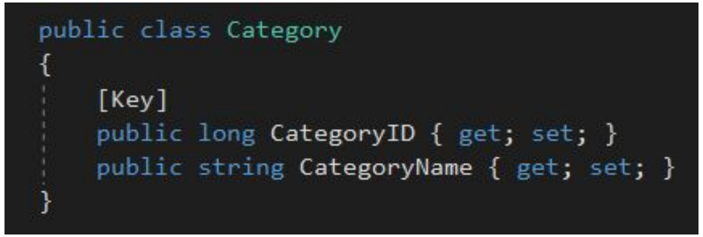
- Let’s modify the previous example using code-first approach.

1) Add Model classes: **Brand**, **Category**, and **Products** in Models Folder.
2) Add data annotation **[Key]** for each primary key
3) Add a class for DbContext with the three DBSets
4) Add ConnectionString in web.config file
5) Build and run the application
6) The database is created based on the model, when first running the application

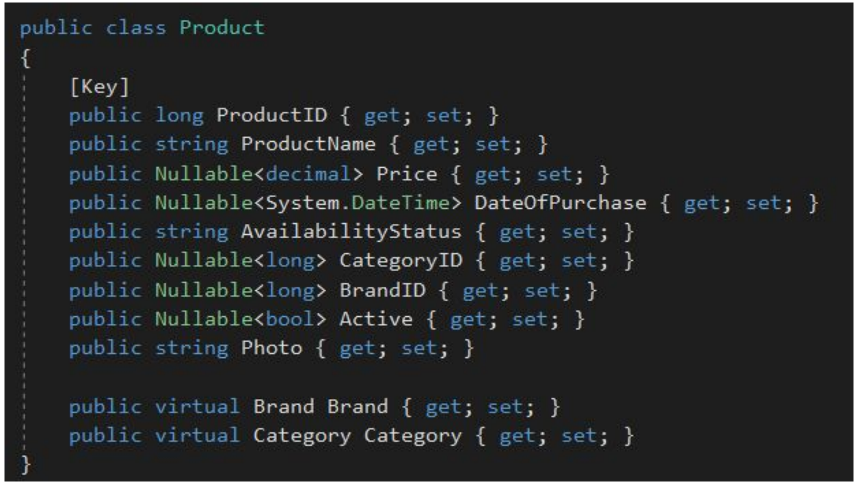
Model classes



```
public class Brand
{
    [Key]
    public long BrandID { get; set; }
    public string BrandName { get; set; }
}
```



```
public class Category
{
    [Key]
    public long CategoryID { get; set; }
    public string CategoryName { get; set; }
}
```



```
public class Product
{
    [Key]
    public long ProductID { get; set; }
    public string ProductName { get; set; }
    public Nullable<decimal> Price { get; set; }
    public Nullable<System.DateTime> DateOfPurchase { get; set; }
    public string AvailabilityStatus { get; set; }
    public Nullable<long> CategoryID { get; set; }
    public Nullable<long> BrandID { get; set; }
    public Nullable<bool> Active { get; set; }
    public string Photo { get; set; }

    public virtual Brand Brand { get; set; }
    public virtual Category Category { get; set; }
}
```

Modify table in Code-First Approach

- Suppose we want to add a column **Quantity** to table **Products**. We are going to do the following:

1) We add the column to Product model class
2) We need to update the database table using one of these approaches:

❖ We delete the database and recreate it

a) Delete database
b) Run the application

Note: In this approach we loose all data from the database

❖ Using Code-First migration

Code-First Migrations

Allows updating of database schema when model changes, without loosing database data.

Code-First Migration are two types:

- Automated Migration
- Code-based Migration

Automated Migration

❖ Updates model changes to database automatically

How to set up automated migrations:

1) Enable Migrations in Package Manager Console

Enable-migrations –EnableAutomaticMigrations:$true

2) Set database initializer in DbContext

Database.SetInitializer(new MigrateDatabaseToLatestVersion<CompanyDbContext, Configuration>());

Example

```
 internal sealed class Configuration :
DbMigrationsConfiguration<EFDbFirstApproachExample.Models.CompanyDbContext>
    {
        public Configuration()
        {
            AutomaticMigrationsEnabled = true;
        }
        protected override void Seed(EFDbFirstApproachExample.Models.CompanyDbContext context)
        {
            context.Brands.AddOrUpdate(new Models.Brand() { BrandID = 1, BrandName = "Samsung" });
            context.Categories.AddOrUpdate(new Models.Category() { CategoryID = 1, CategoryName = "Electronics" });
            context.Products.AddOrUpdate(new Models.Product() { ProductID = 1, ProductName = "Samsung Galaxy Mobile",
CategoryID = 1, DateOfPurchase = DateTime.Now, Active = true, BrandID = 1, Photo = null, Price = 10000, AvailabilityStatus
= "InStock" });
        }
    }
```

Code-based migrations

- Model changes to databases are made based on some automatic-generated classes

How to set up code-based migrations:

1) Enable migrations in Package Manager Console

   Enable-Migrations

2) Add migration class

   Add-migration InitialData

   A class called InitialData.cs will be generated

3) Update database

   update-database -verbose

InititalData.cs

Up Method:

```
CreateTable(
  "dbo.Brands",
  c => new
     {
         BrandID = c.Long(nullable: false, identity: true),
         BrandName = c.String(),
     })
  .PrimaryKey(t => t.BrandID);……………
```

InitialData.cs

```
public override void Down()
    {
        DropForeignKey("dbo.Products", "CategoryID", "dbo.Categories");
        DropForeignKey("dbo.Products", "BrandID", "dbo.Brands");
        DropIndex("dbo.Products", new[] { "BrandID" });
        DropIndex("dbo.Products", new[] { "CategoryID" });
        DropTable("dbo.Products");
        DropTable("dbo.Categories");
        DropTable("dbo.Brands");
    }
```

Adding Quantity column

```
public partial class QuantityColumn : DbMigration
    {
        public override void Up()
        {
            AddColumn("dbo.Products", "Quantity", c => c.Decimal(precision: 18, scale: 2));
        }

        public override void Down()
        {
            DropColumn("dbo.Products", "Quantity");
        }
    }
```

Default conventions in Code-First Approach

- TableName=>Model class + “s” or “es” ***Ex: “Student” class->”Students” table***
- Schema name=> “dbo” ***Ex: “Student” class-> “dbo.Students” table***
- Property name=>Column name ***Ex: “StudentName” property->”StudentName” column***
- Property data type => Column Data Type ***Ex: “string StudentName” -> “StudentName varchar(max) column***
- Primary Key => “Id” ***Ex: “Id” property -> “Id” primary key column***

Overriding default conventions

To override default conventions we use data annotations

- Table
- Column
- Key

Example

```
[Table("Products", Schema ="dbo")]
  public class Product
  {
     [Key]
     public long ProductID { get; set; }
     public string ProductName { get; set; }
     public Nullable<decimal> Price { get; set; }
     [Column("DateOfPurchase", TypeName = "datetime")]
     public Nullable<System.DateTime> DOP { get; set; }
     public string AvailabilityStatus { get; set; }
     public Nullable<long> CategoryID { get; set; }
     public Nullable<long> BrandID { get; set; }
     public Nullable<bool> Active { get; set; }
     public string Photo { get; set; }

     public Nullable<decimal> Quantity { get; set; }

     public virtual Brand Brand { get; set; }
     public virtual Category Category { get; set; }
  }
```

Entity Framework Practice

Change the solution using Entity Framework Code-First approach.

Assignment

Add the solution to Git source control

1. Create a Git account using Github.com
2. Download and install Desktop for Github (desktop.github.com)
3. Create a repository using the server account at Github.com
4. Clone the repository locally

QUESTIONS