ORCHID INTERNATIONAL College

(Affiliated to Tribhuvan University)

A Lab report on NET Centric Computing

Automobile database system

Lab Report No. 03

Submitted by:

Abhinna Ojha

BSc. CSIT, VI

20788/075

Submitted to:

Mr. Dipendra KM

Department of CSIT

Table of Contents

1. Introduction

1.1. Entity Framework

The Entity Framework is a set of technologies in ADO.NET that supports the development of data-oriented software applications. Architects and developers of data-oriented applications have typically struggled with the need to achieve two very different objectives. They must model the entities, relationships, and logic of the business problems they are solving, and they must also work with the data engines used to store and retrieve the data. The data can span multiple storage systems, each with its own protocols; even applications that work with a single storage system must balance the requirements of the storage system against the requirements of writing efficient and maintainable application code. This problem is generally referred to as the "object–relational impedance mismatch".

1.2. Object Relational Model

Object–relational mapping is a programming technique for converting data between type systems using object-oriented programming languages. This creates, in effect, a "virtual object database" that can be used from within the programming language. There are both free and commercial packages available that perform object–relational mapping, although some programmers opt to construct their own ORM tools. In object-oriented programming, data-management tasks act on objects that are almost always non-scalar values.

2. Model

The Model is the part of MVC which implements the domain logic. In simple terms, this logic is used to handle the data passed between the database and the user interface (UI). The Model is known as domain object or domain entity. The domain objects are stored under the Models folder in ASP.NET. The domain model represents the application perspective for the data to be handled whereas a view model is required to produce the engine that generates the View.

3. Database Provider

A database provider is a framework, driver, or object library which enables your web application to send SQL statements to a database and receive data. Sometimes the provider is a class library in your web app's language (here are a few for the .NET framework, for example). In other cases, it may be a type of ODBC driver which you can access using the ODBC standard. A database provider is said to be written with packages like PL/SQL or any other database package.

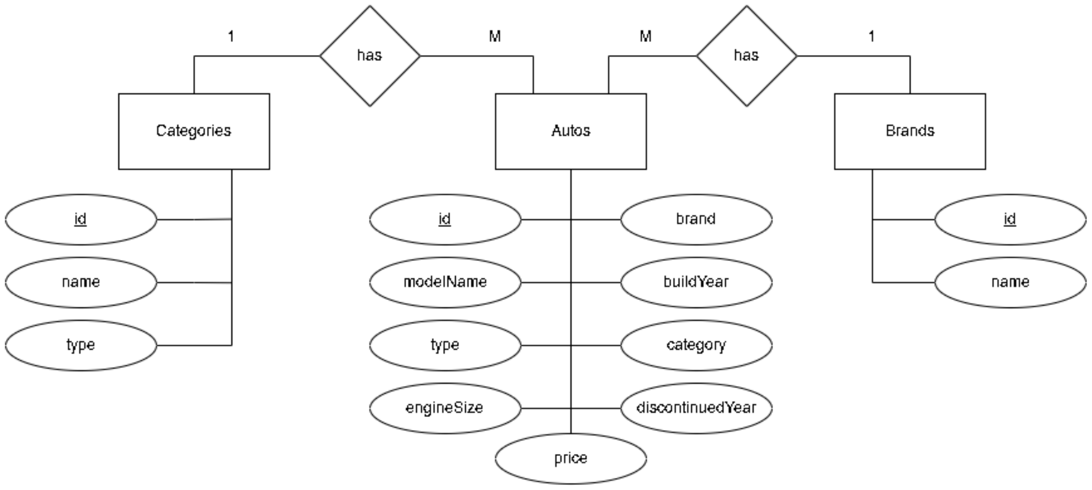
4. DbContext class

A DbContext instance represents a combination of the Unit Of Work and Repository patterns such that it can be used to query from a database and group together changes that will then be written back to the store as a unit. DbContext is conceptually similar to ObjectContext. DbContext is usually used with a derived type that contains DbSet<TEntity> properties for the root entities of the model. When using the Code First approach, the DbSet<TEntity> properties on the derived context are used to build a model by convention.

5. Model validation

When we developed any type of application such as web-based or desktop based, in both the application one is the key part of development is Data Validation or Form Validation. These validations always ensure us that user fills the correct form of data into the application so that we can process that data and save for future use. Model state represents errors that come from two subsystems: model binding and model validation. Errors that originate from model binding are generally data conversion errors. For example, an "x" is entered in an integer field. Model validation occurs after model binding and reports errors where data doesn't conform to business rules. For example, a 0 is entered in a field that expects a rating between 1 and 5. Both model binding and model validation occur before the execution of a controller action or a Razor Pages handler method. For web apps, it's the app's responsibility to inspect ModelState.IsValid and react appropriately.

6. ER diagram



7. Coding

7.1. Auto model

```
namespace automobile_database_system.Models
{
   public class Auto
   {
                    [Key]
                    [Required]
                    public int Id {get; set;}

                    [Required]
                    public string Brand { get; set;}

                    [Required]
                    public string ModelName { get; set;}

                    [Required]
                    public int BuildYear { get; set;}

                    [Required]
                    public int VehicleType { get; set; }

                    [Required]
                    public string Category { get; set;}

                    [Required]
                    public float EngineSize { get; set;}
```

```
            public int DiscontinuedYear { get; set;}

            [Required]
            public float Price { get; set;}
        }
}
```

7.2. Brand model

```
namespace automobile_database_system.Models
{
    public class Brand
    {
        [Key]
        [Required]
        public int Id { get; set; }

        [Required]
        public string BrandName { get; set; }
    }
}
```

7.3. Category model

```
namespace automobile_database_system.Models
{
    public class Category
    {
        [Key]
        [Required]
        public int Id { get; set; }

        [Required]
        public string Name { get; set; }

        [Required]
        public string Type { get; set; }

    }
}
```

7.4. DbContext

```
using Microsoft.EntityFrameworkCore;

namespace automobile_database_system.Models
{
    public class VehicleDatabaseContext:DbContext
    {
        public        VehicleDatabaseContext(DbContextOptions<VehicleDatabaseContext>
options):base(options)
        {

        }

        public DbSet<User> Users { get; set; }
        public DbSet<Category> Categories { get; set; }
        public DbSet<Brand> Brands { get; set; }
        public DbSet<Auto> Autos { get; set; }
    }
}
```

8. Model migration

8.1. Commands

8.1.1. add-migration migration_name

creates a new migration file

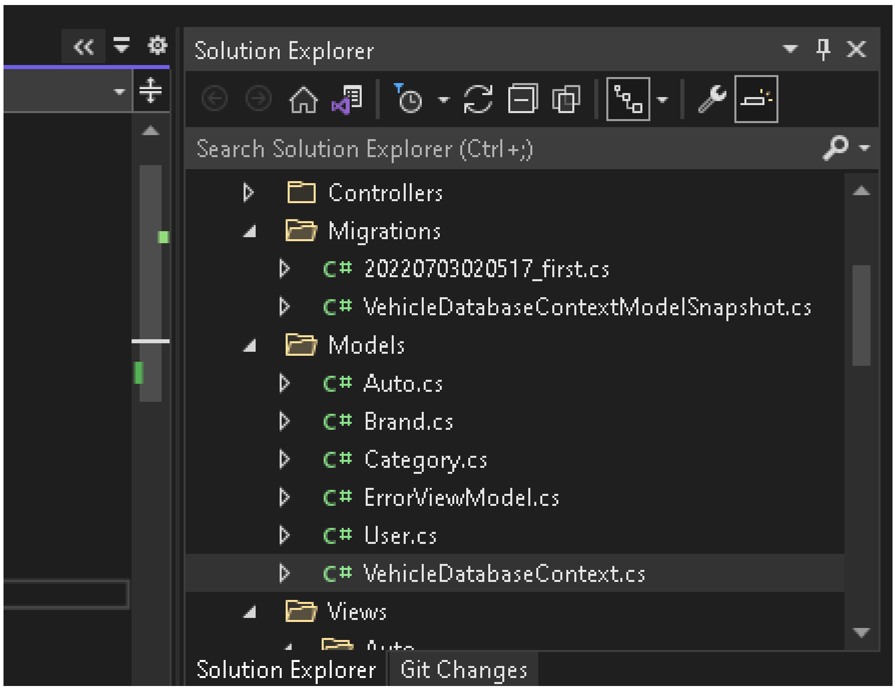
8.1.2. update-database

updates database tables

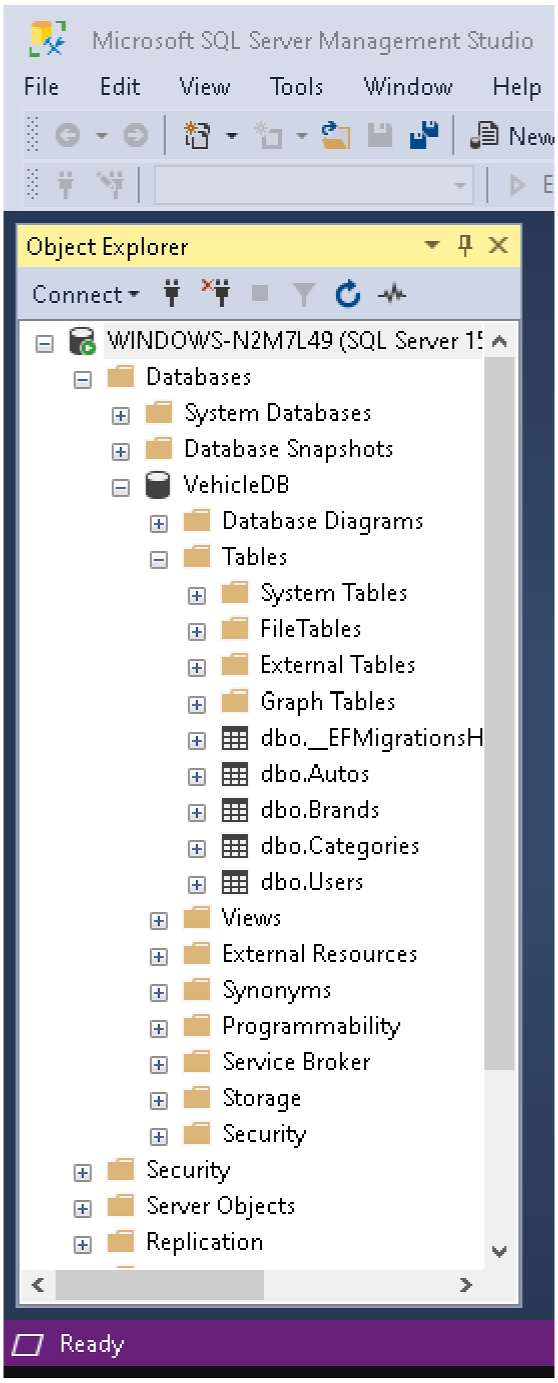
8.2. Migration file code

```
using Microsoft.EntityFrameworkCore.Migrations;
#nullable disable
namespace automobile_database_system.Migrations
{
    public partial class first : Migration
    {
        protected override void Up(MigrationBuilder migrationBuilder)
        {
            migrationBuilder.CreateTable(
                name: "Autos",
                columns: table => new
                {
                    Id = table.Column<int>(type: "int", nullable: false)
                        .Annotation("SqlServer:Identity", "1, 1"),
                    Brand = table.Column<string>(type: "nvarchar(max)", nullable: false),
                    ModelName    =    table.Column<string>(type:    "nvarchar(max)",    nullable:
false),
                    BuildYear = table.Column<int>(type: "int", nullable: false),
                    VehicleType = table.Column<int>(type: "int", nullable: false),
                    Category = table.Column<string>(type: "nvarchar(max)", nullable: false),
                    EngineSize = table.Column<float>(type: "real", nullable: false),
                    DiscontinuedYear = table.Column<int>(type: "int", nullable: false),
                    Price = table.Column<float>(type: "real", nullable: false)
                },
                constraints: table =>
                {
                    table.PrimaryKey("PK_Autos", x => x.Id);
                });

            migrationBuilder.CreateTable(
                name: "Brands",
                columns: table => new
                {
                    Id = table.Column<int>(type: "int", nullable: false)
                        .Annotation("SqlServer:Identity", "1, 1"),
                    BrandName = table.Column<string>(type: "nvarchar(max)", nullable:
false)
                },
                constraints: table =>
                {
                    table.PrimaryKey("PK_Brands", x => x.Id);
                });

            migrationBuilder.CreateTable(
                name: "Categories",
                columns: table => new
                {
                    Id = table.Column<int>(type: "int", nullable: false)
                        .Annotation("SqlServer:Identity", "1, 1"),
                    Name = table.Column<string>(type: "nvarchar(max)", nullable: false),
                    Type = table.Column<string>(type: "nvarchar(max)", nullable: false)
                },
                constraints: table =>
                {
                    table.PrimaryKey("PK_Categories", x => x.Id);
                });

            migrationBuilder.CreateTable(
                name: "Users",
                columns: table => new
                {
                    Id = table.Column<int>(type: "int", nullable: false)
                        .Annotation("SqlServer:Identity", "1, 1"),
                    Username = table.Column<string>(type: "nvarchar(max)", nullable: false),
                    Firstname = table.Column<string>(type: "nvarchar(max)", nullable: false),
                    Middlename = table.Column<string>(type: "nvarchar(max)", nullable:
false),
                    Lastname = table.Column<string>(type: "nvarchar(max)", nullable: false),
                    Email = table.Column<string>(type: "nvarchar(max)", nullable: false),
                    Password = table.Column<string>(type: "nvarchar(max)", nullable: false)
                },
                constraints: table =>
                {
                    table.PrimaryKey("PK_Users", x => x.Id);
                });
        }

        protected override void Down(MigrationBuilder migrationBuilder)
        {
            migrationBuilder.DropTable(
                name: "Autos");

            migrationBuilder.DropTable(
                name: "Brands");

            migrationBuilder.DropTable(
                name: "Categories");

            migrationBuilder.DropTable(
                name: "Users");
        }
    }
}
```

9. Output



```
Solution Explorer
Search Solution Explorer (Ctrl+;)
▷ Controllers
◢ Migrations
    ▷ C# 20220703020517_first.cs
    ▷ C# VehicleDatabaseContextModelSnapshot.cs
◢ Models
    ▷ C# Auto.cs
    ▷ C# Brand.cs
    ▷ C# Category.cs
    ▷ C# ErrorViewModel.cs
    ▷ C# User.cs
    ▷ C# VehicleDatabaseContext.cs
◢ Views
Solution Explorer   Git Changes
```



10. Conclusion

In this lab session, we learnt about routing models, model validation, migrations, and so on. We also did implementation of these concepts. This lab session focused primarily on models, database connection, model validation, migrations, and entity framework. This lab session focused on Entity Framework and using it to make database using code first approach.