Modeling a Simple Database with EF Core

Objective: by the end of this activity, you will be able to apply data modeling techniques to create a simple database using EF Core in a .NET application.

Step 1: Install Required Tools and Create a New Console Application

Set up your environment, install necessary dependencies, and create a new console application for your EF Core project.

Instructions:

1. Install EF Core tools globally.
2. Create a new console application named EFCoreModelApp and navigate to the project directory.
3. Install the EF Core SQLite and tools packages.
4. Run the application to ensure the setup is correct.

Step 2: Create and Configure Entity Classes

Define the Employee and Department entity classes that will represent the tables in your database.

Instructions:

1. Create a folder named Models in your project directory.
2. In the Models folder, create a class for Employee to represent employee records.
3. Create a class for Department to represent department records.
4. Ensure that Employee includes a navigation property for its associated Department.

Step 3: Set Up DbContext

Create the HRDbContext class to manage database connections and relationships between entities.

Instructions:

1. In the root of your project, create a file named HRDbContext.cs.
2. Configure the HRDbContext to include DbSet properties for Employee and Department.

3. Set up a SQLite database connection in the OnConfiguring method.
4. Define the relationships between Employee and Department in the OnModelCreating method.
5. Seed initial data for employees and departments.

Step 4: Add and Apply Migrations

Create the database schema and apply it to your SQLite database.

Instructions:

1. Add an initial migration to capture the current model state.
2. Apply the migration to create the database and tables.

Step 5: Test the Application

Write and run a program to test CRUD operations on the database.

Instructions:

1. Modify the Program.cs file to retrieve and display employee data, including their department names.
2. Add a query to display employees belonging to the HR department.
3. Add functionality to create and save a new employee record.
4. Run the application to verify that the database operations work correctly.

Department.cs:

```
namespace EFCoreModelApp.Models;

public class Department
{
    public int DepartmentId { get; set; }
    public string Name { get; set; } = string.Empty;

    public ICollection<Employee> Employees { get; set; } = [];
}
```

Employee.cs:

```
namespace EFCoreModelApp.Models;

public class Employee
{
    public int EmployeeId { get; set; }
    public string Name { get; set; } = string.Empty;
    public string Position { get; set; } = string.Empty;

    public int DepartmentId { get; set; }

    public Department Department { get; set; } = null!;
}
```

HRDbContext.cs:

```
using Microsoft.EntityFrameworkCore;
using EFCoreModelApp.Models;

namespace EFCoreModelApp.Data;

public class HRDbContext : DbContext
{
    public DbSet<Employee> Employees { get; set; } = null!;
    public DbSet<Department> Departments { get; set; } = null!;

    protected override void OnConfiguring(DbContextOptionsBuilder
optionsBuilder)
    {
        optionsBuilder.UseSqlite("Data Source=hr.db");
    }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        // One-to-many relationship
        modelBuilder.Entity<Employee>()
            .HasOne(e => e.Department)
            .WithMany(d => d.Employees)
            .HasForeignKey(e => e.DepartmentId);

        // Seed initial data
        modelBuilder.Entity<Department>().HasData(
            new Department { DepartmentId = 1, Name = "HR" },
            new Department { DepartmentId = 2, Name = "IT" }
        );

        modelBuilder.Entity<Employee>().HasData(
            new Employee { EmployeeId = 1, Name = "Alice", Position =
"HR Manager", DepartmentId = 1 },
            new Employee { EmployeeId = 2, Name = "Bob", Position =
"Developer", DepartmentId = 2 }
        );
    }
}
```

Program.cs:

```
using EFCoreModelApp.Data;
using EFCoreModelApp.Models;

using var context = new HRDbContext();

context.Database.EnsureCreated();

void PrintTable<T>(IEnumerable<T> items, string title)
{
    Console.WriteLine($"\n{title}");
    Console.WriteLine(new string('-', 100));

    var props = typeof(T).GetProperties();

    foreach (var prop in props)
        Console.Write($"{prop.Name,-30}");
    Console.WriteLine();
    Console.WriteLine(new string('-', 100));

    foreach (var item in items)
    {
        foreach (var prop in props)
        {
            var value = prop.GetValue(item) ?? "";
            Console.Write($"{value,-30}");
        }
        Console.WriteLine();
    }

    Console.WriteLine(new string('-', 100));
}

var employees = context.Employees
    .Select(e => new { e.EmployeeId, e.Name, e.Position, Department =
e.Department.Name })
    .ToList();

PrintTable(employees, "All Employees");

var hrEmployees = context.Employees
    .Where(e => e.Department.Name == "HR")
    .Select(e => new { e.EmployeeId, e.Name, e.Position, Department =
e.Department.Name })
```

```
    .ToList();

PrintTable(hrEmployees, "Employees in HR Department");

if (!context.Employees.Any(e => e.Name == "Charlie"))
{
    var newEmployee = new Employee
    {
        Name = "Charlie",
        Position = "System Administrator",
        DepartmentId = 2
    };

    context.Employees.Add(newEmployee);
    context.SaveChanges();
    Console.WriteLine("\nAdded new employee: Charlie (System
Administrator, IT).");
}
else
{
    Console.WriteLine("\nCharlie already exists. Skipping insert.");
}

var updatedEmployees = context.Employees
    .Select(e => new { e.EmployeeId, e.Name, e.Position, Department =
e.Department.Name })
    .ToList();

PrintTable(updatedEmployees, "Updated Employees List");

var departments = context.Departments
    .Select(d => new { d.DepartmentId, d.Name })
    .ToList();

PrintTable(departments, "Departments");
```