Agenda: CRUD Operations using Entity Framework

- Basic CRUD Operations using Entity Framework.
- Entity Framework Core Code First Model.
- Database Seeding.
- Reverse Engineer Model Classes.
- Separation of work using BO Classes.

Basic CRUD Operations using Entity Framework Core

1. Create a New ASP.NET Application

File → New Project → Visual C# →.NET Core → ASP.NET Core Web Application → OK

Ensure the options **.NET Core** and **ASP.NET Core** are selected in the drop down lists

2. Create Model classes

- Define a context and entity classes that make up your model.
- Right-click on the Models folder and select Add -> Class...

```
public class Department
{
   [Key]
   public int Id { get; set; }
   public string DeptName { get; set; }
   public bool IsActive { get; set; }
   public ICollection<Employee> Employees { get; set; }
}
public class Employee
{
   [Key]
   public int Id { get; set; }
   [ForeignKey("Department")]
   public int FKDeptId { get; set; }
   public string EmpName { get; set; }
   public decimal Salary { get; set; }
   public bool IsActive { get; set; }
   public Department Department { get; set; }
}
```

Note: By default, EF Core interprets a property that's named **ID** or **classnameID** as the primary key.

```
[DatabaseGenerated(DatabaseGeneratedOption.None)]
```

The **DatabaseGenerated** attribute allows the app to specify the primary key rather than having the DB generate it.

3. Build the Project and ensure there are **no compilation errors**.
4. In **Solution Explorer**, right click on the *Controllers Folder* → **Add** → **New Scaffolded Item**. → **MVC Controller with views, using Entity Framework** → Add
5. Select the Model class = "Employee", Data context class: Click +, New data context type: **OrganizationContext** → Add
6. **Examine the code generated.**

Examine the context generated context class: Data/DemoDbContext.cs

```
public class DemoDbContext : DbContext
{
    public DemoDbContext (DbContextOptions<DemoDbContext> options)
        : base(options)
    {
    }
    public DbSet<DataAccessUsingEF.Models.Employee> Employee { get; set; }
    public DbSet<DataAccessUsingEF.Models.Department> Department { get; set; }
}
```

Examine the context registered with dependency injection:

Open **Startup.cs** and look at the **ConfigureService**

```
services.AddDbContext<DemoDbContext>(options =>
    options.UseSqlServer(Configuration.GetConnectionString("DemoDbContext")));
```

Examine the appsettings.json file

```
"ConnectionStrings": {
  "DemoDbContext": "Server=(localdb)\\mssqllocaldb;Database=DemoDbContext-a522a8a2-2980-4de2-b158-
381de1e28908;Trusted_Connection=True;MultipleActiveResultSets=true"
 }
```

7. Edit the Code in Main as below

```
public static void Main(string[] args)
{
  var host = CreateWebHostBuilder(args).Build();
  using (var scope = host.Services.CreateScope())
  {
    var services = scope.ServiceProvider;
    try
    {
      var context = services.GetRequiredService<DemoDbContext>();
      context.Database.EnsureCreated();
    }
    catch (Exception ex)
    {
      var logger = services.GetRequiredService<ILogger<Program>>();
      logger.LogError(ex, "An error occurred creating the DB.");
    }
  }
  host.Run();
}
```

- **EnsureCreated** ensures that the database for the context exists. If it exists, no action is taken. If it does not exist, then the database and all its schema are created.
- EnsureCreated does not use migrations to create the database. A database that is created with EnsureCreated cannot be later updated using migrations.
- EnsureCreated is called on app start, which allows us with the following workflow:
  - Delete the DB.
  - Change the DB schema (for example, add an EmailAddress field).
  - Run the app.
  - EnsureCreated creates a DB with the EmailAddress column.
- EnsureCreated is convenient early in development when the schema is rapidly evolving. Later in the tutorial the DB is deleted and migrations are used.

8. Build the application

Some things to be aware of when writing asynchronous code that uses EF Core:

- Only statements that cause queries or commands to be sent to the DB are executed asynchronously. That includes, `ToListAsync`, `SingleOrDefaultAsync`, `FirstOrDefaultAsync`, and `SaveChangesAsync`. It doesn't include statements that just change an `IQueryable` , such as `var students = context.Students.Where(s => s.LastName == "Sam")` .

- An EF Core context isn't thread safe: don't try to do multiple operations in parallel.
- To take advantage of the performance benefits of async code, verify that library packages (such as for paging) use async if they call EF Core methods that send queries to the DB.

Add code to initialize the DB with test data

9. Add New class DbInitializer to the Project.

```
public static class DbInitializer
{
  public static void Initialize(DemoContext context)
  {
    // context.Database.EnsureCreated();
    if (!context.Department.Any())
    {
      var departments = new Department[]
      {
        new Department(){DeptName = "Development"},
        new Department(){DeptName = "Training"}
      };
      foreach (Department dept in departments)
        context.Department.Add(dept);
      context.SaveChanges();
    }
    // Look for any students.
    if (!context.Employee.Any())
      {
        var employees = new Employee[]
        {
          new Employee(){EmpName="Sandeep", FKDeptId=1, Salary=100000, IsActive=true },
          new Employee(){EmpName="Rahul",FKDeptId=2, Salary=100000, IsActive=true},
          new Employee(){EmpName="Naveen",FKDeptId=1, Salary=50000, IsActive=true}
        };
```

```
            foreach (Employee emp in employees)
                context.Employee.Add(emp);
            context.SaveChanges();
        }
    }
}
```

10. Edit Main.cs and add the following line after EnsureCreated call

```
context.Database.EnsureCreated();
DbInitializer.Initialize(context);
```

Migration Tools

For early development, **EnsureCreated** was used. In this tutorial, migrations are used. It has the following limitations:

- Bypasses migrations and creates the DB and schema.
- Doesn't create a migrations table.
- Cann*ot* be used with migrations.
- Is designed for testing or rapid prototyping where the DB is dropped and re-created frequently.

11. Remove the following line from Main

    `//context.Database.EnsureCreated();`

The following steps use **migrations** to create a database.

12. Package Manager Console → Drop Database
13. Create the Initial Migration and Update the database.

    Open Tools → NuGet Package Manager → **Package Manager Console**

    - Run `Add-Migration InitialCreate` to scaffold a migration to create the initial set of tables for your model.

      If you receive an error

    - Run `Update-Database` to apply the new migration to the database. This command creates the database before applying migrations.

Examine the Up and Down methods:

Note that Migrations calls the **Up** method to implement the data model changes for a migration. When you enter a command to roll back the update, migrations calls the **Down** method.

14. Run the app and verify the DB is seeded.

Database.Migrate()

Applies any pending migrations for the context to the database. Will create the database if it does not already exist.

Applying migrations in production

- We recommend production apps should **not** call **Database.Migrate** at application startup.
- **Migrate** shouldn't be called from an app in server farm. For example, if the app has been cloud deployed with scale-out (multiple instances of the app are running).
- Database migration should be done as part of deployment, and in a controlled way. Production database migration approaches include:
  - Using migrations to create SQL scripts and using the SQL scripts in deployment.
  - Running **dotnet ef database update** from a controlled environment.
- EF Core uses the **__MigrationsHistory** table to see if any migrations need to run. If the DB is up-to-date, no migration is run.

Reverse Engineer Model Classes

Reverse engineering to create an Entity Framework model based on an existing database.

1. In Sql Server Create New database name “Organization” with two tables, Department and Employee as below

```sql
CREATE TABLE Department(
        PKDeptId int Primary Key IDENTITY(1,1),
        DeptName varchar(50) NOT NULL,
        IsActive bit NOT NULL);
```

```sql
CREATE TABLE Employee(
        PKEmpId int Primary Key IDENTITY(1,1),
        EmpName varchar(50) NOT NULL,
        Salary money NOT NULL,
        IsActive bit NOT NULL,
        FKDeptId int NOT NULL,
CONSTRAINT FK_Employee_Department FOREIGN KEY(FKDeptId)
REFERENCES Department(PKDeptId));
```

Tools –> NuGet Package Manager –> Package Manager Console

```
Scaffold-DbContext "Server=.\sqlexpress;Database=DemoDb;Trusted_Connection=True;"
Microsoft.EntityFrameworkCore.SqlServer -OutputDir Models
```

Using BO Classes

1. Create a New Folder, Name = BO

2. Under BO Folder, add the EmployeeBO class to the project

```
public class EmployeeBO
 {
        OrganizationContext db;
    public EmployeeBO(OrganizationContext db)
    {
       this.db = db;
    }
    public IEnumerable<Employee> GetAll(int deptId = 0)
    {
       IEnumerable<Employee> qry = db.Employees.Include(e=>e.Department);
       if (deptId != 0)
          qry = qry.Where(emp => emp.FKDeptId == deptId);
       return qry.ToList();
    }
    public Employee GetById(int? id)
    {
       return db.Employees.Find(id);
    }
    public Employee Add(Employee entity)
    {
       db.Employees.Add(entity);
       db.SaveChanges();
       return entity;
    }
    public void Update(Employee entity)
    {
        db.Entry<Employee>(entity).State = EntityState.Modified;
        db.SaveChanges();
    }
    public void Delete(int? id)
    {
        Employee entity = db.Employees.Find(id);
        db.Remove(entity);
        db.SaveChanges();
    }
    public void Dispose()
    {
        db.Dispose();
    }
}
```

3. Under BO Folder, add the DepartmentBO class to the project

```
public class DepartmentBO
{
    OrganizationContext db;
    public DepartmentBO(OrganizationContext db)
    {
        this.db = db;
    }
    public IEnumerable<Department> GetAll()
    {
        IEnumerable<Department> qry = db.Departments;
        return qry.ToList();
    }
    public Department GetById(int? id)
    {
        return db.Departments.Find(id);
    }
    public Department Add(Department entity)
    {
        db.Departments.Add(entity);
```

```
            db.SaveChanges();
            return entity;
        }
        public void Update(Department entity)
        {
            db.Entry(entity).State = EntityState.Modified;
            db.SaveChanges();
        }
        public void Delete(int? id)
        {
            Department entity = db.Departments.Find(id);
            db.Remove(entity);
            db.SaveChanges();
        }
        public void Dispose()
        {
            db.Dispose();
        }
    }
```

4. Edit the code of EmployeesContoller as below:

EmployeesController constructor accepting one parameter that context object we are injecting from the service.

```
public class EmployeesController : Controller
    {
        EmployeeBO objEmployeeBO;
        DepartmentBO objDepartmentBO;
        public EmployeesController(OrganizationContext context)
        {
            objEmployeeBO = new EmployeeBO(context);
            objDepartmentBO = new DepartmentBO(context);
        }

        // GET: Employees
        public IActionResult Index()
        {
```

```
        // var data = _context.Employees.Include(e => e.Department).ToList();
        var data = objEmployeeBO.GetAll();
        return View(data);
    }

    // GET: Employees/Details/5
    public IActionResult Details(int? id)
    {
        if (id == null)
        {
            return new BadRequestResult();
        }

        var employee = objEmployeeBO.GetById(id);
        if (employee == null)
        {
            return NotFound();
        }

        return View(employee);
    }

    // GET: Employees/Create
    public IActionResult Create()
    {
        ViewBag.Depts = new SelectList(objDepartmentBO.GetAll(), "PKDeptId", "DeptName");
        return View();
    }

    // POST: Employees/Create
    // To protect from overposting attacks, please enable the specific properties you want to bind to, for
    // more details see http://go.microsoft.com/fwlink/?LinkId=317598.
    [HttpPost]
    [ValidateAntiForgeryToken]
    public IActionResult Create(Employee employee)
    {
      if (ModelState.IsValid)
      {
        objEmployeeBO.Add(employee);
        return RedirectToAction("Index");
      }
      ViewBag.Depts = new SelectList(objDepartmentBO.GetAll(), "PKDeptId", "DeptName");
      return View(employee);
    }

    // GET: Employees/Edit/5
    public IActionResult Edit(int? id)
    {
      if (id == null)
      {
        return new BadRequestResult();
      }

      var employee = objEmployeeBO.GetById(id);
      if (employee == null)
      {
        return NotFound();
      }
      ViewBag.Depts = new SelectList(objDepartmentBO.GetAll(), "PKDeptId", "DeptName",
employee.FKDeptId);
      return View(employee);
    }

    // POST: Employees/Edit/5
    // To protect from overposting attacks, please enable the specific properties you want to bind to, for
    // more details see http://go.microsoft.com/fwlink/?LinkId=317598.
    [HttpPost]
    [ValidateAntiForgeryToken]
    public IActionResult Edit(int id, [Bind("PKEmpId,EmpName,Salary,IsActive,FKDeptId")] Employee employee)
    {
            if (id != employee.PKEmpId)
            {
                return new BadRequestResult();
            }

            if (ModelState.IsValid)
            {
                objEmployeeBO.Update(employee);
                return RedirectToAction("Index");
            }
            ViewBag.Depts = new SelectList(objDepartmentBO.GetAll(), "PKDeptId", "DeptName",
employee.FKDeptId);
            return View(employee);
        }

        // GET: Employees/Delete/5
        public IActionResult Delete(int? id)
        {
            if (id == null)
            {
                return new BadRequestResult();
            }

            var employee = objEmployeeBO.GetById(id);
            if (employee == null)
            {
                return NotFound();
            }

            return View(employee);
        }

        // POST: Employees/Delete/5
        [HttpPost, ActionName("Delete")]
        [ValidateAntiForgeryToken]
        public IActionResult DeleteConfirmed(int id)
        {
            objEmployeeBO.Delete(id);
            return RedirectToAction("Index");
        }
    }
```

5. Build and run the application.