Outline:

- Code First and Migrations (EFCore 8)
- Data annotation
- Fluent API
- Configuration (Columns, Mapping)
- Execute Delete and Update
- stored procedures
- Loading (Lazy, Eager, Explicit)

Mapping

EF Core Conventions

1. one-to-one
2. one-to-many
3. man-to-many

Fluent API

⇒ Fluent API configuration has the highest precedence and will override conventions and data annotations.

- Configure entity properties (e.g., primary keys, column names, data types).
- Define relationships between entities (e.g., one-to-one, one-to-many, many-to-many).

1. Entity Configuration

- `HasKey()`: Configures the primary key.

```
modelBuilder.Entity<Students>()
            .HasKey(a => a.StudentId);
```

- `Property()`: Configures a property (e.g., data type, max length, required).

```
modelBuilder.Entity<Employee>()
            .Property(a=>a.Name)
            .HasColumnType("varchar")
            .HasMaxLength(100)
            .IsRequired();
```

2. Relationship Configuration

- `HasOne()`: Configures a one-to-one or one-to-many relationship.

```
modelBuilder.Entity<Employee>()
            .HasOne(E => E.address)
            .WithOne(a => a.Employee)
            .HasForeignKey<Employee>(e => e.AddressId);
```

- `HasMany()`: Configures a one-to-many or many-to-many relationship.

```
modelBuilder.Entity<Department>()
            .HasMany(d => d.Employees)
```

```
            .WithOne(e => e.department)
            .HasForeignKey(e => e.DepartmentId);
```

- `UsingEntity()`: Configures the join table for many-to-many relationships.

3. Database-Specific Configuration

- `ToTable()`: Specifies the table name for an entity.
- `HasDefaultValue()`: Sets a default value for a column.

Data Annotation

⇒ These attributes are not only used in Entity Framework but they can also be used with ASP.NET MVC or data controls.

```
[key]
public int S_ID; // primary key

[Required]
[MaxLength(150)]
[DeafultValue("stree")]
[Column(TypeName = "VarChar")]
public string address;
```

Data Seeding

⇒ Adding Default or initial data

```
modelBuilder.Entity<Department>().HasData(
    new Department { Id = 1, Name = "HR" },
    new Department { Id = 2, Name = "IT" }
);
```

Execute Delete & Update

⇒ only start from Ef Core 7

```
contx.Departments.ExecuteUpdate(a=>a.SetProperty(d=>d.Name,"name")); //update name for all column

contx.Addresses.Where(a=>a.Id==1).ExecuteDelete(); // delete row with id 1

contx.Employees.ExecuteDelete(); // delete all data in Employees (Bulk)
```

Eager Loading

⇒ By default, when you retrieve an object, its navigation properties are not automatically loaded.

⇒ To enable Eager Loading, you use the `Include` keyword.

Example

```
var employees = contx.Employees
                .Include(e => e.Department).ToList();
```

Generated Sql Query

```
SELECT [e].[Id], [e].[AddressId], [e].[DepartmentId], [e].[Name], [e].[Salary], [d].[Id], [d].[Name]
FROM [Employees] AS [e]
INNER JOIN [Departments] AS [d] ON [e].[DepartmentId] = [d].[Id]
```

Explicit Loading

⇒ load data when needed by using `contx.Entry`

```
var emp1 = contx.Employees
               .FirstOrDefault(a => a.Id == 6);

var emp2 = contx.Employees
               .FirstOrDefault(a => a.Id == 7);

contx.Entry(emp1).Reference(a => a.department).Query();
contx.Entry(emp2).Reference(a => a.department).Load();
```

Lazy Loading

⇒ to enable Lazy loading in EF Core

- Install proxies Package
- enable lazy loading in dbcontext
- make all navigation property "virtual"

References

- Migrations Overview - EF Core | Microsoft Learn