**EF Core 8.0 (Labs 1–5)  
  
Lab 1:   
 Understanding ORM with a Retail Inventory System  
  
ORM:**

ORM is a technique that connects the rich objects of an application (like C# classes) to tables in a relational database, automating the conversion (mapping) between them.

**How it works in C#:**

· Each class in C# corresponds to a table in the database.

· Each property in the class maps to a column.

· Relationships like one-to-many, many-to-many are represented using navigation properties and foreign keys. **EF Core vs EF Framework:**

| **Feature** | **EF Core** | **EF Framework (EF6)** |
| --- | --- | --- |
| Platform | Cross-platform (.NET Core, .NET 6/7/8) | Windows-only |
| Performance | Lightweight and faster | Heavier |
| Modern Features | LINQ, async, compiled queries | Limited async support |
| Maturity | Still evolving | More mature/stable |
| Migration Support | Yes | Yes |
| Usage | Recommended for new apps | Legacy support only |

**EF Core 8.0 Features:**

JSON Column Mapping:

Store complex types like lists/dictionaries as JSON in a single column.

Compiled Models:

Faster startup time; reduces runtime overhead by pre-compiling model metadata.

Interceptors:

Hook into database operations (e.g., logging, modifying commands).

Bulk Operations:

Improved support for batching INSERT/UPDATE/DELETE.

Create a .NET Console App:

bash

CopyEdit

dotnet new console -n RetailInventorycd RetailInventory

Install EF Core Packages for SQL Server:

bash

CopyEdit

dotnet add package Microsoft.EntityFrameworkCore.SqlServer  
 **Lab 2:   
 Setting Up the Database Context for a Retail Store**

**Create Models:**

public class Category

{

public int Id { get; set; }

public string Name { get; set; }

public List<Product> Products { get; set; }

}

public class Product

{

public int Id { get; set; }

public string Name { get; set; }

public decimal Price { get; set; }

public int CategoryId { get; set; }

public Category Category { get; set; }

}

Create AppDbContext:

using Microsoft.EntityFrameworkCore;

public class AppDbContext : DbContext

{

public DbSet<Product> Products { get; set; }

public DbSet<Category> Categories { get; set; }

protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)

{

optionsBuilder.UseSqlServer("Server=localhost;Database=RetailDb;Trusted\_Connection=True;");

}

}

**Optional (For ASP.NET Core): Use appsettings.json:**

In ASP.NET Core apps, it’s better to configure the connection string in appsettings.json:

json

CopyEdit

{

"ConnectionStrings": {

"DefaultConnection": "Server=localhost;Database=RetailDb;Trusted\_Connection=True;"

}}

Then configure it in Program.cs or Startup.cs:

csharp

CopyEdit

builder.Services.AddDbContext<AppDbContext>(options =>

options.UseSqlServer(builder.Configuration.GetConnectionString("DefaultConnection")));  
  
**Lab 3:**

**Using EF Core CLI to Create and Apply Migrations**

**Install EF Core CLI:**

dotnet tool install --global dotnet-ef

Create the Initial Migration:

dotnet ef migrations add InitialCreate  
Apply the Migration and Create the Database:

dotnet ef database update

**Verify in SQL Server:**

Open SQL Server Management Studio (SSMS) or Azure Data Studio.

Connect to the server instance mentioned in your connection string.

Check if the database exists (e.g., RetailDb).

Expand Tables – you should see:

Categories

Products

\_\_EFMigrationsHistory (EF Core internal)

**Lab 4:  
 Inserting Initial Data into the Database**

**Insert Data in Program.cs:**

Program.cs:

using System;

using System.Threading.Tasks;

using Microsoft.EntityFrameworkCore;

class Program

{

static async Task Main(string[] args)

{

using var context = new AppDbContext();

var electronics = new Category { Name = "Electronics" };

var groceries = new Category { Name = "Groceries" };

await context.Categories.AddRangeAsync(electronics, groceries);

var product1 = new Product { Name = "Laptop", Price = 75000, Category = electronics };

var product2 = new Product { Name = "Rice Bag", Price = 1200, Category = groceries };

await context.Products.AddRangeAsync(product1, product2);

await context.SaveChangesAsync();

Console.WriteLine("Initial data inserted successfully!");

}

}

**Run the Application:**

dotnet run

Initial data inserted successfully!

**Verify in SQL Server:**

SELECT \* FROM Categories;

SELECT \* FROM Products; **Lab 5:**

**Retrieving Data from the Database**

Program.cs:

using System;

using System.Threading.Tasks;

using Microsoft.EntityFrameworkCore;

class Program

{

static async Task Main(string[] args)

{

using var context = new AppDbContext();

Console.WriteLine("\n All Products:");

var products = await context.Products.ToListAsync();

foreach (var p in products)

Console.WriteLine($"{p.Name} - ₹{p.Price}");

Console.WriteLine("\n Find Product by ID (1):");

var product = await context.Products.FindAsync(1);

Console.WriteLine($"Found: {product?.Name}");

Console.WriteLine("\n First Product with Price > ₹50000:");

var expensive = await context.Products.FirstOrDefaultAsync(p => p.Price > 50000);

Console.WriteLine($"Expensive: {expensive?.Name}");

}

}

**Output:**

**A screen shot of a computer

AI-generated content may be incorrect.**

**Lab 6:**

**Updating and Deleting Records**

Program.cs:

using System;

using System.Threading.Tasks;

using Microsoft.EntityFrameworkCore;

class Program

{

static async Task Main(string[] args)

{

using var context = new AppDbContext();

var product = await context.Products.FirstOrDefaultAsync(p => p.Name == "Laptop");

if (product != null)

{

Console.WriteLine($"Before Update: {product.Name} - ₹{product.Price}");

product.Price = 70000;

await context.SaveChangesAsync();

Console.WriteLine($"After Update: {product.Name} - ₹{product.Price}");

}

var toDelete = await context.Products.FirstOrDefaultAsync(p => p.Name == "Rice Bag");

if (toDelete != null)

{

context.Products.Remove(toDelete);

await context.SaveChangesAsync();

Console.WriteLine($"Deleted: {toDelete.Name}");

}

Console.WriteLine("\n📦 Updated Product List:");

var products = await context.Products.ToListAsync();

foreach (var p in products)

Console.WriteLine($"{p.Name} - ₹{p.Price}");

}

}

**Output:**

**A screenshot of a computer screen

AI-generated content may be incorrect.**

**Lab 7:**

**Writing Queries with LINQ**

Program.cs

using System;

using System.Threading.Tasks;

using System.Linq;

using Microsoft.EntityFrameworkCore;

class Program

{

static async Task Main(string[] args)

{

using var context = new AppDbContext();

var filtered = await context.Products

.Where(p => p.Price > 1000)

.OrderByDescending(p => p.Price)

.ToListAsync();

Console.WriteLine("\n📊 Filtered & Sorted Products (Price > ₹1000):");

foreach (var p in filtered)

Console.WriteLine($"{p.Name} - ₹{p.Price}");

var productDTOs = await context.Products

.Select(p => new { p.Name, p.Price })

.ToListAsync();

Console.WriteLine("\n📄 Product DTOs (Name + Price only):");

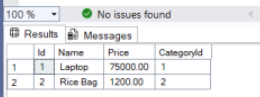
foreach (var dto in productDTOs)

Console.WriteLine($"{dto.Name} - ₹{dto.Price}");

}

}

**Output:**

****