**Lab 1: Understanding ORM with a Retail Inventory System**

**Objective**

To understand what Object-Relational Mapping (ORM) is and how Entity Framework Core (EF Core) helps bridge the gap between C# objects and SQL Server relational tables.

**1. What is ORM?**

Object-Relational Mapping (ORM) is a technique that connects the rich objects of an application to tables in a relational database. ORM maps classes in C# to database tables, and properties to table columns.

Benefits of ORM:

• Improved productivity

• Easier maintenance of code and data models

• Abstraction from raw SQL queries

**2. EF Core vs EF Framework**

EF Core:

• Cross-platform, lightweight, and modern

• Supports LINQ, async queries, compiled queries

EF Framework (EF6):

• Windows-only and more mature

• Less flexible and not cross-platform

**3. EF Core 8.0 Features**

• JSON column mapping

• Improved performance with compiled models

• Support for interceptors and efficient bulk operations

**4. Step-by-Step Implementation**

Step 1: Create a new .NET Console App

Command: dotnet new console -n RetailInventory

Step 2: Install EF Core packages

Commands:

• dotnet add package Microsoft.EntityFrameworkCore.SqlServer

• dotnet add package Microsoft.EntityFrameworkCore.Design

Step 3: Create Models

Product.cs

public class Product  
{  
 public int ProductId { get; set; }  
 public string Name { get; set; } = string.Empty;  
 public int Stock { get; set; }  
 public Category Category { get; set; } = null!;  
}

Category.cs

public class Category  
{  
 public int CategoryId { get; set; }  
 public string Name { get; set; } = string.Empty;  
 public List<Product> Products { get; set; } = new();  
}

Step 4: Create AppDbContext.cs

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=(localdb)\MSSQLLocalDB;Database=RetailDb;Trusted\_Connection=True;");  
 }  
}

Step 5: Update Program.cs

class Program  
{  
 static void Main(string[] args)  
 {  
 using var context = new AppDbContext();  
  
 if (!context.Categories.Any())  
 {  
 var electronics = new Category { Name = "Electronics" };  
 var groceries = new Category { Name = "Groceries" };  
  
 context.Categories.AddRange(electronics, groceries);  
  
 context.Products.AddRange(  
 new Product { Name = "Laptop", Stock = 10, Category = electronics },  
 new Product { Name = "Banana", Stock = 50, Category = groceries }  
 );  
  
 context.SaveChanges();  
 }  
  
 foreach (var product in context.Products.Include(p => p.Category))  
 {  
 Console.WriteLine($"{product.Name} - Stock: {product.Stock} - Category: {product.Category?.Name}");  
 }  
 }  
}

Step 6: Add Migration and Update Database

• dotnet ef migrations add InitialCreate

• dotnet ef database update

**5. Output**

The following output confirms successful ORM operation:

