# **Lab 1: Understanding ORM - Codes**

## **NuGet Package Installation Commands**

### **Package Manager Console Commands:**

powershell

# Install SQL Server packageInstall-Package Microsoft.EntityFrameworkCore.SqlServer# Install Design package for migrationsInstall-Package Microsoft.EntityFrameworkCore.Design# Install Tools package for CLI commandsInstall-Package Microsoft.EntityFrameworkCore.Tools

### **Visual Studio Solution Explorer Structure:**

RetailInventory/├── Dependencies/│ ├── Packages/│ │ ├── Microsoft.EntityFrameworkCore.SqlServer│ │ ├── Microsoft.EntityFrameworkCore.Design│ │ └── Microsoft.EntityFrameworkCore.Tools├── Models/│ ├── Category.cs│ └── Product.cs├── Data/│ └── AppDbContext.cs├── Migrations/│ └── (Generated migration files)├── Program.cs└── RetailInventory.csproj

### **Project File (.csproj) - Auto-generated after package installation:**

xml

<Project Sdk="Microsoft.NET.Sdk"> <PropertyGroup> <OutputType>Exe</OutputType> <TargetFramework>net8.0</TargetFramework> <Nullable>enable</Nullable> </PropertyGroup> <ItemGroup> <PackageReference Include="Microsoft.EntityFrameworkCore.Design" Version="8.0.0" /> <PackageReference Include="Microsoft.EntityFrameworkCore.SqlServer" Version="8.0.0" /> <PackageReference Include="Microsoft.EntityFrameworkCore.Tools" Version="8.0.0" /> </ItemGroup></Project>

## **ORM Concept Explanation Code Example**

### Traditional ADO.NET Approach (What ORM Replaces):

csharp

// WITHOUT ORM - Traditional ADO.NET (Complex)using (var connection = new SqlConnection(connectionString)){ connection.Open(); var command = new SqlCommand("SELECT \* FROM Products WHERE CategoryId = @categoryId", connection); command.Parameters.AddWithValue("@categoryId", 1); using (var reader = command.ExecuteReader()) { while (reader.Read()) { var product = new Product { Id = reader.GetInt32("Id"), Name = reader.GetString("Name"), Price = reader.GetDecimal("Price") }; // Manual mapping... } }}

### **With EF Core ORM (Simple):**

csharp

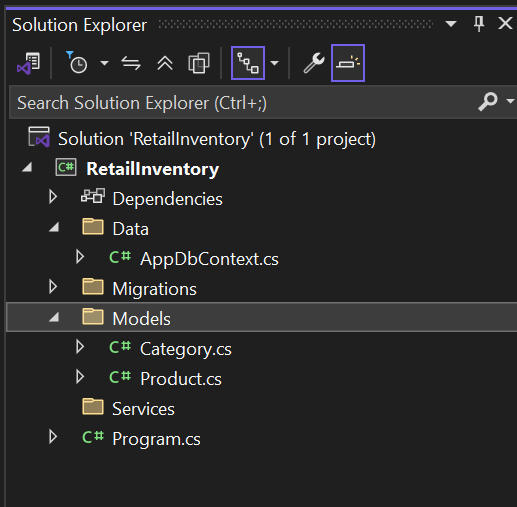
// WITH EF Core ORM - Simple and Cleanusing var context = new AppDbContext();var products = await context.Products .Where(p => p.CategoryId == 1) .ToListAsync();

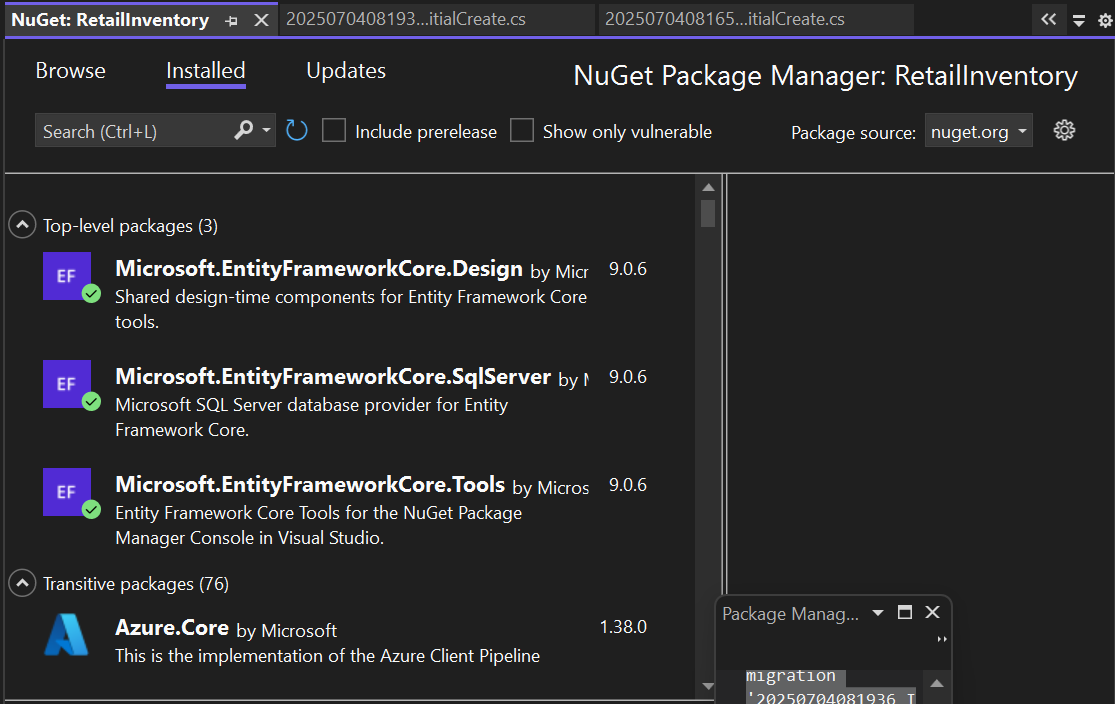
## EF Core 8.0 Features Mentioned

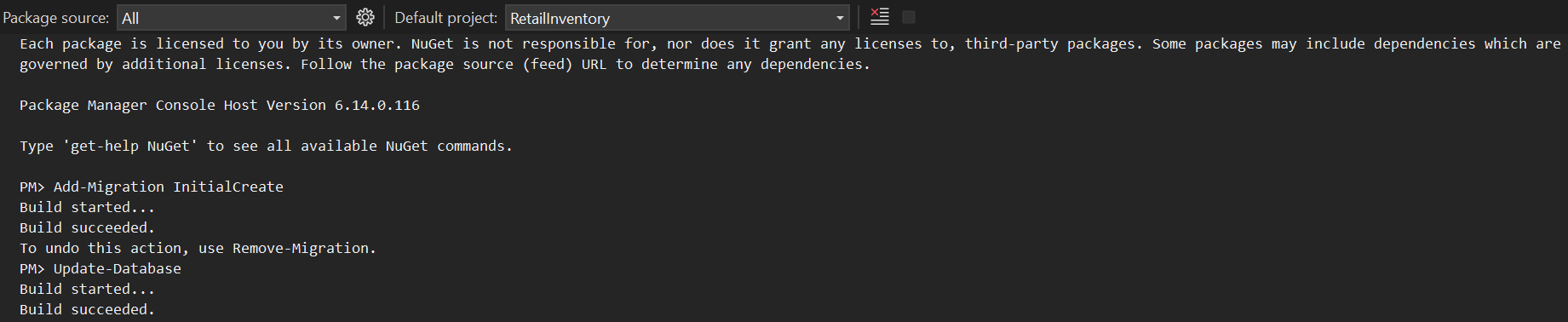
csharp

// Example of EF Core 8.0 features that will be used in this project:// 1. Async queriesvar products = await context.Products.ToListAsync();// 2. LINQ supportvar expensiveProducts = context.Products.Where(p => p.Price > 50000);// 3. Navigation propertiesvar productsWithCategories = context.Products.Include(p => p.Category);// 4. Compiled models (better performance)// 5. Improved bulk operations// 6. Better interceptors

*Output*







**Lab 2: Setting Up Database Context - Codes**

**Connection String Configuration**

Option 1: Direct in DbContext (Used in our project)

protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)

{

// Connection string for SQL Server LocalDB

optionsBuilder.UseSqlServer(

@"Server=(localdb)\mssqllocaldb;Database=RetailInventoryDB;Trusted\_Connection=true;MultipleActiveResultSets=true"

);

}

Option 2: Using appsettings.json (Alternative approach)

{

"ConnectionStrings": {

"DefaultConnection": "Server=(localdb)\\mssqllocaldb;Database=RetailInventoryDB;Trusted\_Connection=true;MultipleActiveResultSets=true"

}

}

Option 3: Dependency Injection (For ASP.NET Core)

// In Program.cs or Startup.cs

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

options.UseSqlServer(builder.Configuration.GetConnectionString("DefaultConnection")));

**Models/Category.cs**

using System.ComponentModel.DataAnnotations;

namespace RetailInventory.Models

{

public class Category

{

public int Id { get; set; }

[Required]

[StringLength(100)]

public string Name { get; set; } = string.Empty;

// Navigation property - One Category has Many Products

public virtual ICollection<Product> Products { get; set; } = new List<Product>();

}

}

**Models/Product.cs**

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

namespace RetailInventory.Models

{

public class Product

{

public int Id { get; set; }

[Required]

[StringLength(200)]

public string Name { get; set; } = string.Empty;

[Column(TypeName = "decimal(18,2)")]

public decimal Price { get; set; }

public int StockQuantity { get; set; } = 0;

// Foreign Key

public int CategoryId { get; set; }

// Navigation property - Many Products belong to One Category

public virtual Category Category { get; set; } = null!;

}

}

**Data/AppDbContext.cs**

using Microsoft.EntityFrameworkCore;

using RetailInventory.Models;

namespace RetailInventory.Data

{

public class AppDbContext : DbContext

{

// DbSet represents tables in database

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

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

protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)

{

// Configure database connection

optionsBuilder.UseSqlServer(

@"Server=(localdb)\mssqllocaldb;Database=RetailInventoryDB;Trusted\_Connection=true;MultipleActiveResultSets=true"

);

}

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

// Configure Product entity

modelBuilder.Entity<Product>(entity =>

{

entity.HasKey(p => p.Id);

entity.Property(p => p.Name).IsRequired().HasMaxLength(200);

entity.Property(p => p.Price).HasColumnType("decimal(18,2)");

// Configure relationship: One Category has Many Products

entity.HasOne(p => p.Category)

.WithMany(c => c.Products)

.HasForeignKey(p => p.CategoryId)

.OnDelete(DeleteBehavior.Cascade);

});

// Configure Category entity

modelBuilder.Entity<Category>(entity =>

{

entity.HasKey(c => c.Id);

entity.Property(c => c.Name).IsRequired().HasMaxLength(100);

});

}

}

}

**Key Concepts Explained in Code**

1. DbContext Inheritance

public class AppDbContext : DbContext // Inherits from EF Core's DbContext

{

// This class represents your database session

}

2. DbSet Properties

public DbSet<Product> Products { get; set; } // Represents Products table

public DbSet<Category> Categories { get; set; } // Represents Categories table

3. Entity Relationships

// One-to-Many relationship configuration

entity.HasOne(p => p.Category) // One Category

.WithMany(c => c.Products) // Many Products

.HasForeignKey(p => p.CategoryId) // Foreign Key

.OnDelete(DeleteBehavior.Cascade); // Delete behavior

4. Data Annotations vs Fluent API

// Data Annotations (in Model classes)

[Required]

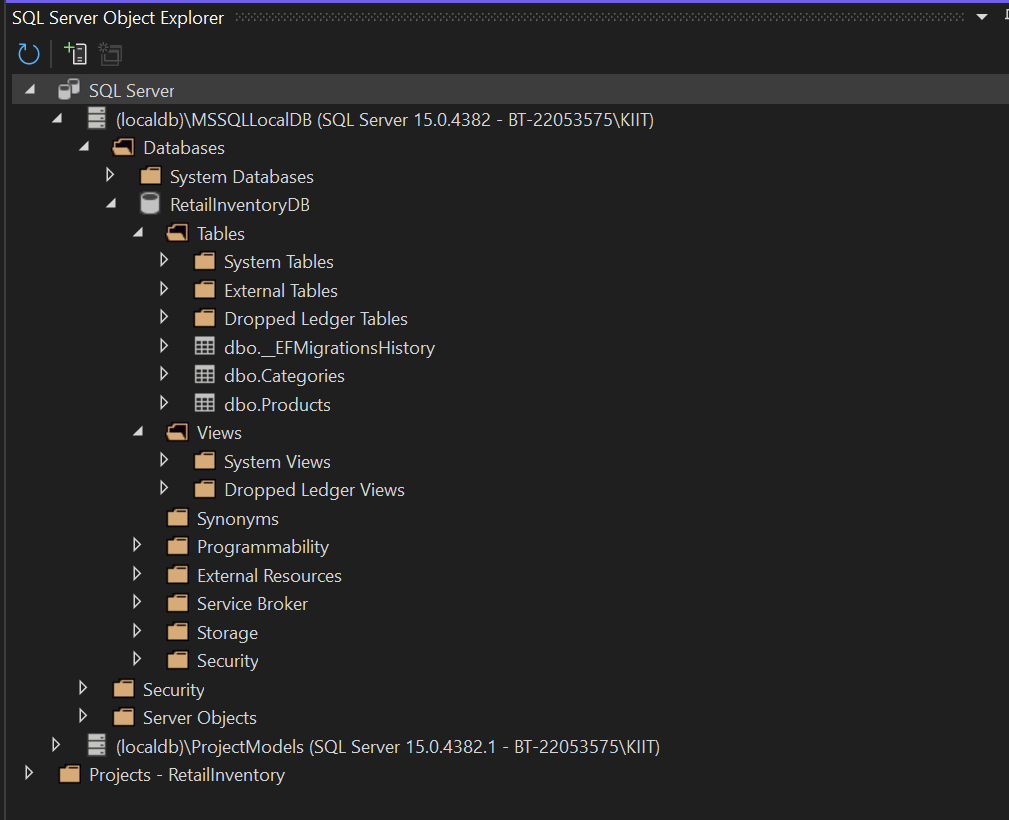
[StringLength(100)]

public string Name { get; set; }

// Fluent API (in OnModelCreating method)

entity.Property(p => p.Name).IsRequired().HasMaxLength(200);

**Outputs**



**Lab 3: Using EF Core CLI for Migrations - Codes**

**Migration Commands**

**Package Manager Console Commands:**

# Install EF Core CLI tools globally (one-time setup)

dotnet tool install --global dotnet-ef

# Create initial migration

Add-Migration InitialCreate

# Apply migration to create database

Update-Database

# View migration history

Get-Migration

# Remove last migration (if needed)

Remove-Migration

# Drop database (if needed)

Drop-Database

**Generated Migration File (InitialCreate.cs)**

**Sample Migration File Content:**

using Microsoft.EntityFrameworkCore.Migrations;

#nullable disable

namespace RetailInventory.Migrations

{

/// <inheritdoc />

public partial class InitialCreate : Migration

{

/// <inheritdoc />

protected override void Up(MigrationBuilder migrationBuilder)

{

// Create Categories table

migrationBuilder.CreateTable(

name: "Categories",

columns: table => new

{

Id = table.Column<int>(type: "int", nullable: false)

.Annotation("SqlServer:Identity", "1, 1"),

Name = table.Column<string>(type: "nvarchar(100)", maxLength: 100, nullable: false)

},

constraints: table =>

{

table.PrimaryKey("PK\_Categories", x => x.Id);

});

// Create Products table

migrationBuilder.CreateTable(

name: "Products",

columns: table => new

{

Id = table.Column<int>(type: "int", nullable: false)

.Annotation("SqlServer:Identity", "1, 1"),

Name = table.Column<string>(type: "nvarchar(200)", maxLength: 200, nullable: false),

Price = table.Column<decimal>(type: "decimal(18,2)", nullable: false),

StockQuantity = table.Column<int>(type: "int", nullable: false),

CategoryId = table.Column<int>(type: "int", nullable: false)

},

constraints: table =>

{

table.PrimaryKey("PK\_Products", x => x.Id);

table.ForeignKey(

name: "FK\_Products\_Categories\_CategoryId",

column: x => x.CategoryId,

principalTable: "Categories",

principalColumn: "Id",

onDelete: ReferentialAction.Cascade);

});

// Create index for foreign key

migrationBuilder.CreateIndex(

name: "IX\_Products\_CategoryId",

table: "Products",

column: "CategoryId");

}

/// <inheritdoc />

protected override void Down(MigrationBuilder migrationBuilder)

{

// Drop tables in reverse order

migrationBuilder.DropTable(name: "Products");

migrationBuilder.DropTable(name: "Categories");

}

}

}

**Model Snapshot File**

AppDbContextModelSnapshot.cs (Auto-generated):

using Microsoft.EntityFrameworkCore;

using Microsoft.EntityFrameworkCore.Infrastructure;

using Microsoft.EntityFrameworkCore.Metadata;

using RetailInventory.Data;

#nullable disable

namespace RetailInventory.Migrations

{

[DbContext(typeof(AppDbContext))]

partial class AppDbContextModelSnapshot : ModelSnapshot

{

protected override void BuildModel(ModelBuilder modelBuilder)

{

modelBuilder

.HasAnnotation("ProductVersion", "8.0.0")

.HasAnnotation("Relational:MaxIdentifierLength", 128);

SqlServerModelBuilderExtensions.UseIdentityColumns(modelBuilder);

modelBuilder.Entity("RetailInventory.Models.Category", b =>

{

b.Property<int>("Id")

.ValueGeneratedOnAdd()

.HasColumnType("int");

SqlServerPropertyBuilderExtensions.UseIdentityColumn(b.Property<int>("Id"));

b.Property<string>("Name")

.IsRequired()

.HasMaxLength(100)

.HasColumnType("nvarchar(100)");

b.HasKey("Id");

b.ToTable("Categories");

});

modelBuilder.Entity("RetailInventory.Models.Product", b =>

{

b.Property<int>("Id")

.ValueGeneratedOnAdd()

.HasColumnType("int");

SqlServerPropertyBuilderExtensions.UseIdentityColumn(b.Property<int>("Id"));

b.Property<int>("CategoryId")

.HasColumnType("int");

b.Property<string>("Name")

.IsRequired()

.HasMaxLength(200)

.HasColumnType("nvarchar(200)");

b.Property<decimal>("Price")

.HasColumnType("decimal(18,2)");

b.Property<int>("StockQuantity")

.HasColumnType("int");

b.HasKey("Id");

b.HasIndex("CategoryId");

b.ToTable("Products");

});

modelBuilder.Entity("RetailInventory.Models.Product", b =>

{

b.HasOne("RetailInventory.Models.Category", "Category")

.WithMany("Products")

.HasForeignKey("CategoryId")

.OnDelete(DeleteBehavior.Cascade)

.IsRequired();

b.Navigation("Category");

});

modelBuilder.Entity("RetailInventory.Models.Category", b =>

{

b.Navigation("Products");

});

}

}

}

**Migration Process Verification**

Check Database Structure (SQL Commands):

-- View created tables

SELECT TABLE\_NAME FROM INFORMATION\_SCHEMA.TABLES WHERE TABLE\_TYPE = 'BASE TABLE'

-- View Categories table structure

SELECT COLUMN\_NAME, DATA\_TYPE, IS\_NULLABLE, CHARACTER\_MAXIMUM\_LENGTH

FROM INFORMATION\_SCHEMA.COLUMNS

WHERE TABLE\_NAME = 'Categories'

-- View Products table structure

SELECT COLUMN\_NAME, DATA\_TYPE, IS\_NULLABLE, CHARACTER\_MAXIMUM\_LENGTH

FROM INFORMATION\_SCHEMA.COLUMNS

WHERE TABLE\_NAME = 'Products'

-- View foreign key relationships

SELECT

FK.CONSTRAINT\_NAME,

FK.TABLE\_NAME,

FK.COLUMN\_NAME,

PK.TABLE\_NAME AS REFERENCED\_TABLE\_NAME,

PK.COLUMN\_NAME AS REFERENCED\_COLUMN\_NAME

FROM INFORMATION\_SCHEMA.REFERENTIAL\_CONSTRAINTS C

INNER JOIN INFORMATION\_SCHEMA.KEY\_COLUMN\_USAGE FK ON C.CONSTRAINT\_NAME = FK.CONSTRAINT\_NAME

INNER JOIN INFORMATION\_SCHEMA.KEY\_COLUMN\_USAGE PK ON C.UNIQUE\_CONSTRAINT\_NAME = PK.CONSTRAINT\_NAME

**Migration Best Practices**

Example of Adding New Migration:

# If you modify your models later, create a new migration

Add-Migration AddProductDescription

# Always review the generated migration before applying

# Then apply it

**Update-Database**

**Example Migration for Model Changes:**

// If you add a new property to Product model:

public string Description { get; set; } = string.Empty;

// The new migration would contain:

migrationBuilder.AddColumn<string>(

name: "Description",

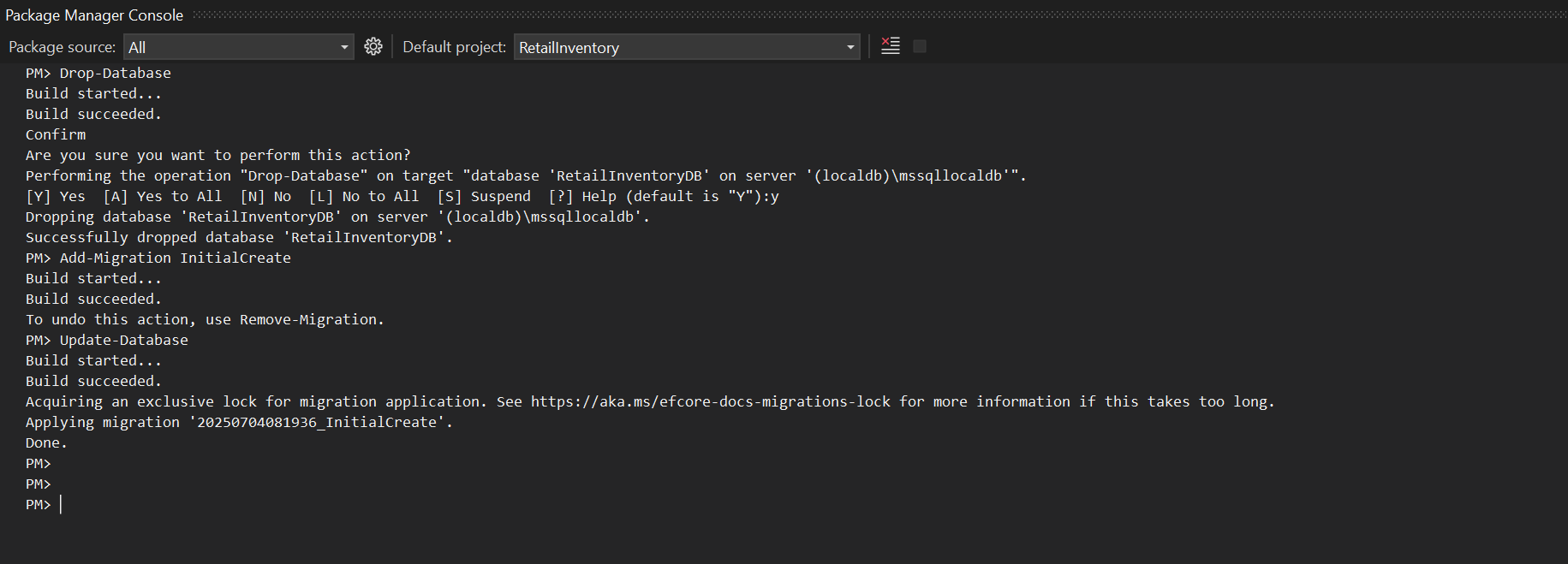
table: "Products",

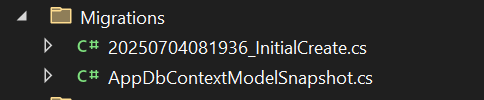
type: "nvarchar(max)",

nullable: false,

defaultValue: "");

**Output**





**Lab 4: Inserting Initial Data - Codes**

**Data Seeding in AppDbContext.cs**

SeedData Method (Add this to AppDbContext.cs):

private void SeedData(ModelBuilder modelBuilder)

{

// Seed Categories

modelBuilder.Entity<Category>().HasData(

new Category { Id = 1, Name = "Electronics" },

new Category { Id = 2, Name = "Groceries" },

new Category { Id = 3, Name = "Clothing" },

new Category { Id = 4, Name = "Books" }

);

// Seed Products

modelBuilder.Entity<Product>().HasData(

new Product { Id = 1, Name = "Laptop", Price = 75000, StockQuantity = 10, CategoryId = 1 },

new Product { Id = 2, Name = "Rice Bag (5kg)", Price = 1200, StockQuantity = 50, CategoryId = 2 },

new Product { Id = 3, Name = "Smartphone", Price = 45000, StockQuantity = 15, CategoryId = 1 },

new Product { Id = 4, Name = "T-Shirt", Price = 800, StockQuantity = 30, CategoryId = 3 },

new Product { Id = 5, Name = "Headphones", Price = 5000, StockQuantity = 20, CategoryId = 1 },

new Product { Id = 6, Name = "Jeans", Price = 2500, StockQuantity = 25, CategoryId = 3 },

new Product { Id = 7, Name = "Programming Book", Price = 1500, StockQuantity = 12, CategoryId = 4 }

);

}

**Complete OnModelCreating Method:**

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

// Configure entities (relationships, constraints, etc.)

// ... entity configurations ...

// Seed initial data

SeedData(modelBuilder);

}

Program.cs - Data Insertion Methods

Add New Product Method:

private static async Task AddNewProduct(AppDbContext context)

{

Console.WriteLine("\n=== ADD NEW PRODUCT ===");

// Show available categories first

var categories = await context.Categories.ToListAsync();

Console.WriteLine("Available Categories:");

foreach (var cat in categories)

{

Console.WriteLine($"{cat.Id}. {cat.Name}");

}

try

{

Console.Write("Product Name: ");

string name = Console.ReadLine() ?? "";

Console.Write("Price: ₹");

decimal price = decimal.Parse(Console.ReadLine() ?? "0");

Console.Write("Stock Quantity: ");

int stock = int.Parse(Console.ReadLine() ?? "0");

Console.Write("Category ID: ");

int categoryId = int.Parse(Console.ReadLine() ?? "0");

// Create new product object

var product = new Product

{

Name = name,

Price = price,

StockQuantity = stock,

CategoryId = categoryId

};

// Add to context using AddAsync

context.Products.Add(product);

// Save changes to database using SaveChangesAsync

await context.SaveChangesAsync();

Console.WriteLine("Product added successfully!");

}

catch (Exception ex)

{

Console.WriteLine($"Error adding product: {ex.Message}");

}

}

**Add New Category Method:**

private static async Task AddNewCategory(AppDbContext context)

{

Console.WriteLine("\n=== ADD NEW CATEGORY ===");

try

{

Console.Write("Category Name: ");

string name = Console.ReadLine() ?? "";

// Create new category object

var category = new Category { Name = name };

// Add to context using AddAsync

context.Categories.Add(category);

// Save changes to database using SaveChangesAsync

await context.SaveChangesAsync();

Console.WriteLine("Category added successfully!");

}

catch (Exception ex)

{

Console.WriteLine($"Error adding category: {ex.Message}");

}

}

Bulk Insert Method (AddRangeAsync):

private static async Task SeedDataIfEmpty(AppDbContext context)

{

// Check if data already exists

if (await context.Categories.AnyAsync())

{

Console.WriteLine("Data already exists in the database.\n");

return;

}

Console.WriteLine("Seeding initial data...");

// Create categories

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

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

var clothing = new Category { Name = "Clothing" };

// Add multiple categories at once using AddRangeAsync

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

// Create products

var products = new List<Product>

{

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

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

new Product { Name = "Smartphone", Price = 45000, Category = electronics },

new Product { Name = "T-Shirt", Price = 800, Category = clothing },

new Product { Name = "Headphones", Price = 5000, Category = electronics }

};

// Add multiple products at once using AddRangeAsync

await context.Products.AddRangeAsync(products);

// Save all changes to database

await context.SaveChangesAsync();

Console.WriteLine("Data seeded successfully!\n");

}

Update Product Stock Method:

private static async Task UpdateProductStock(AppDbContext context)

{

Console.WriteLine("\n=== UPDATE PRODUCT STOCK ===");

try

{

Console.Write("Product ID: ");

int id = int.Parse(Console.ReadLine() ?? "0");

// Find product by ID

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

if (product != null)

{

Console.WriteLine($"Current stock for {product.Name}: {product.StockQuantity}");

Console.Write("New stock quantity: ");

int newStock = int.Parse(Console.ReadLine() ?? "0");

// Update property

product.StockQuantity = newStock;

// Save changes (EF Core tracks changes automatically)

await context.SaveChangesAsync();

Console.WriteLine("Stock updated successfully!");

}

else

{

Console.WriteLine("Product not found.");

}

}

catch (Exception ex)

{

Console.WriteLine($"Error updating stock: {ex.Message}");

}

}

Delete Product Method:

private static async Task DeleteProduct(AppDbContext context)

{

Console.WriteLine("\n=== DELETE PRODUCT ===");

try

{

Console.Write("Product ID to delete: ");

int id = int.Parse(Console.ReadLine() ?? "0");

// Find product by ID

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

if (product != null)

{

Console.WriteLine($"Are you sure you want to delete '{product.Name}'? (y/n)");

string confirm = Console.ReadLine() ?? "";

if (confirm.ToLower() == "y")

{

// Remove from context

context.Products.Remove(product);

// Save changes to database

await context.SaveChangesAsync();

Console.WriteLine("Product deleted successfully!");

}

else

{

Console.WriteLine("Delete cancelled.");

}

}

else

{

Console.WriteLine("Product not found.");

}

}

catch (Exception ex)

{

Console.WriteLine($"Error deleting product: {ex.Message}");

}

}

**Key EF Core Methods Used for Data Insertion**

1. Adding Single Entity:

context.Products.Add(product); // Add single product

await context.SaveChangesAsync(); // Save to database

2. Adding Multiple Entities:

await context.Products.AddRangeAsync(products); // Add multiple products

await context.SaveChangesAsync(); // Save to database

3. Updating Entity:

var product = await context.Products.FindAsync(id); // Find entity

product.StockQuantity = newStock; // Modify property

await context.SaveChangesAsync(); // Save changes

4. Deleting Entity:

var product = await context.Products.FindAsync(id); // Find entity

context.Products.Remove(product); // Mark for deletion

await context.SaveChangesAsync(); // Save changes

Database Seeding vs Runtime Insertion

Database Seeding (OnModelCreating):

// This happens during migration - data is inserted when database is created

modelBuilder.Entity<Product>().HasData(

new Product { Id = 1, Name = "Laptop", Price = 75000, CategoryId = 1 }

);

Runtime Insertion (Program.cs):

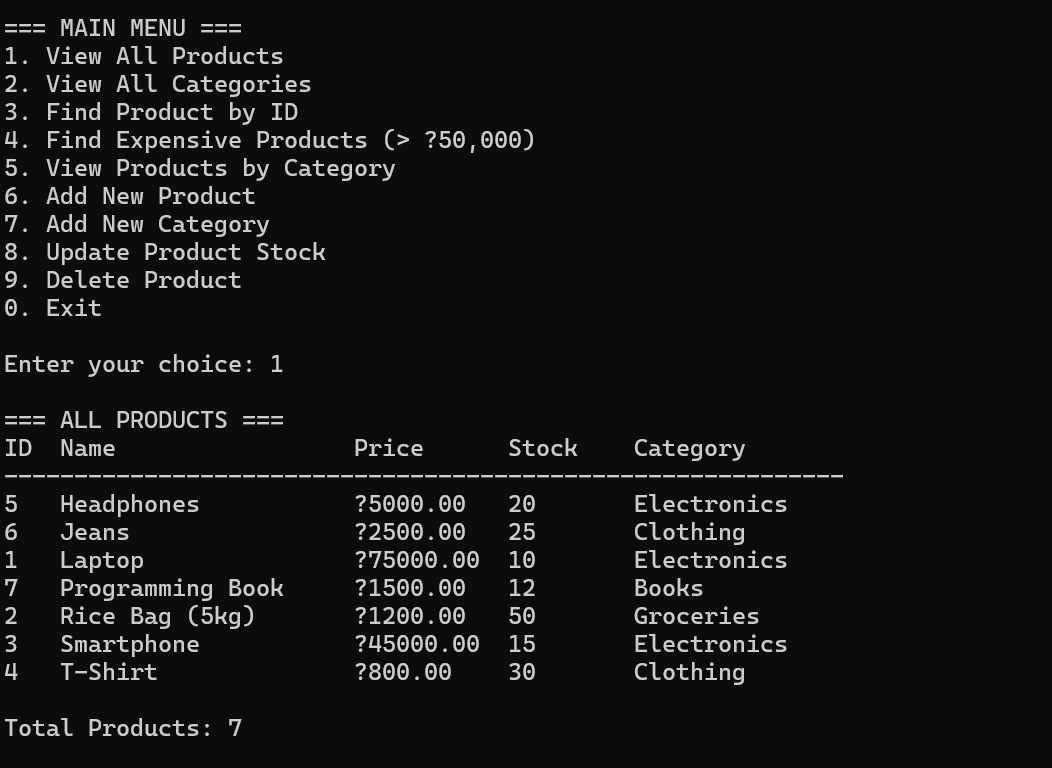
// This happens when application runs - data is inserted dynamically

var product = new Product { Name = "Laptop", Price = 75000, CategoryId = 1 };

context.Products.Add(product);

await context.SaveChangesAsync();

**Output**



# Lab 5: Retrieving Data from Database - Codes

## Program.cs - Data Retrieval Methods

### Display All Products Method (ToListAsync):

private static async Task DisplayAllProducts(AppDbContext context)

{

Console.WriteLine("\n=== ALL PRODUCTS ===");

// ToListAsync() - Retrieves all products from database

var products = await context.Products

.Include(p => p.Category) // Include related Category data

.OrderBy(p => p.Name) // Sort by name

.ToListAsync(); // Execute query and get list

if (products.Any())

{

Console.WriteLine($"{"ID",-3} {"Name",-20} {"Price",-10} {"Stock",-8} {"Category",-15}");

Console.WriteLine(new string('-', 60));

foreach (var product in products)

{

Console.WriteLine($"{product.Id,-3} {product.Name,-20} ₹{product.Price,-9} {product.StockQuantity,-8} {product.Category.Name,-15}");

}

Console.WriteLine($"\nTotal Products: {products.Count}");

}

else

{

Console.WriteLine("No products found.");

}

}

### Find Product by ID Method (FindAsync):

private static async Task FindProductById(AppDbContext context)

{

Console.Write("\nEnter Product ID: ");

if (int.TryParse(Console.ReadLine(), out int id))

{

// FindAsync() - Finds entity by primary key

var product = await context.Products

.Include(p => p.Category) // Include related Category

.FirstOrDefaultAsync(p => p.Id == id); // Alternative to FindAsync with Include

if (product != null)

{

Console.WriteLine($"\nFound Product:");

Console.WriteLine($"ID: {product.Id}");

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

Console.WriteLine($"Price: ₹{product.Price}");

Console.WriteLine($"Stock: {product.StockQuantity}");

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

}

else

{

Console.WriteLine("Product not found.");

}

}

else

{

Console.WriteLine("Invalid ID format.");

}

}

### Find Expensive Products Method (FirstOrDefaultAsync with Where):

private static async Task FindExpensiveProducts(AppDbContext context)

{

Console.WriteLine("\n=== EXPENSIVE PRODUCTS (> ₹50,000) ===");

// FirstOrDefaultAsync() with Where clause - Find products matching condition

var expensiveProducts = await context.Products

.Include(p => p.Category) // Include related Category

.Where(p => p.Price > 50000) // Filter condition

.OrderByDescending(p => p.Price) // Sort by price (highest first)

.ToListAsync(); // Get all matching products

if (expensiveProducts.Any())

{

foreach (var product in expensiveProducts)

{

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

}

}

else

{

Console.WriteLine("No expensive products found.");

}

}

### View Products by Category Method (Include and GroupBy):

private static async Task ViewProductsByCategory(AppDbContext context)

{

Console.WriteLine("\n=== PRODUCTS BY CATEGORY ===");

// Include() - Load related data (Navigation Properties)

var categories = await context.Categories

.Include(c => c.Products) // Include all products for each category

.OrderBy(

