# Entity Framework Core Exercise Solutions

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

1. What is ORM?

*ans:* ORM (Object –Relational Mapping) is a programming technique that allows to interact with a **relational database** (like MySQL, SQL Server, PostgreSQL) using **object-oriented programming** (like C# or Java) — without writing raw SQL queries.

1. How ORM maps C# classes to Database Tables?

*ans:* ORM maps each **C# class** to a **database table**, and each **property** in the class to a **column** in that table. For example, a class Product with properties Id, Name, and Price will map to a table Products with the same columns.

1. *Benefits: Productivity, maintainability, and abstraction from SQL.*

*ans*:

1. **Productivity**  
    ORM allows developers to focus on writing C# code instead of SQL. Common operations like inserting or querying data can be done with simple method calls.
2. **Maintainability**  
    Since database logic is written in code (not scattered SQL scripts), it’s easier to refactor, debug, and maintain applications over time.
3. **Abstraction from SQL**  
    ORM hides the complexity of SQL behind object-oriented code. Developers don't need to know complex SQL queries to perform database operations.

## Lab 2: Setting Up Database Context for a Retail Store

1. Create Models:
   1. Models:

|  |
| --- |
| //Category Model using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;    namespace RetailInventory.models  {  public class Category  {  public int Id { get; set; }  public string Name { get; set; }  public List<Product> Products { get; set; }  }  }  //Products Model  using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;    namespace RetailInventory.models  {  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; }  }  } |

* 1. AppDbContext:

|  |
| --- |
| using Microsoft.EntityFrameworkCore;  using RetailInventory.models;  using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;    namespace RetailInventory.Database  {  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;");  }  }  } |

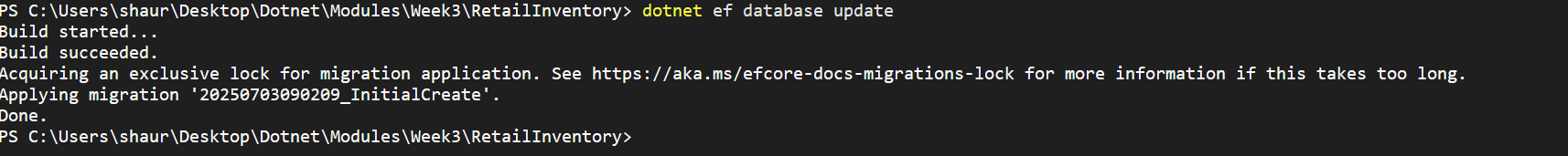
*Output:*

*Installing tools and updating the database is in Lab –3, so are the outputs.*

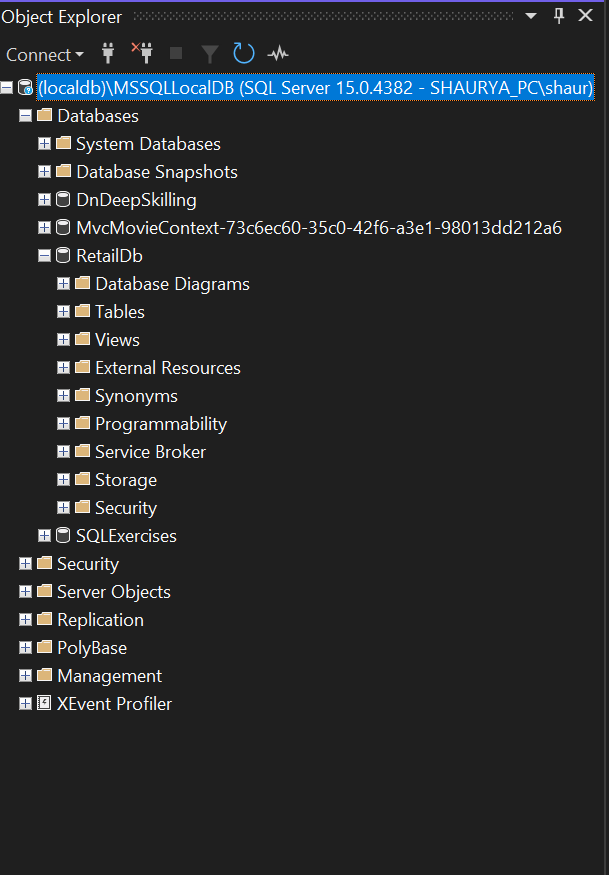
## Lab – 3: Using EF Core CLI to Create and Apply Migrations

*Code: The code implementation for the output below is done in Lab –2.*

*Output – 1:*



*Output – 2:*



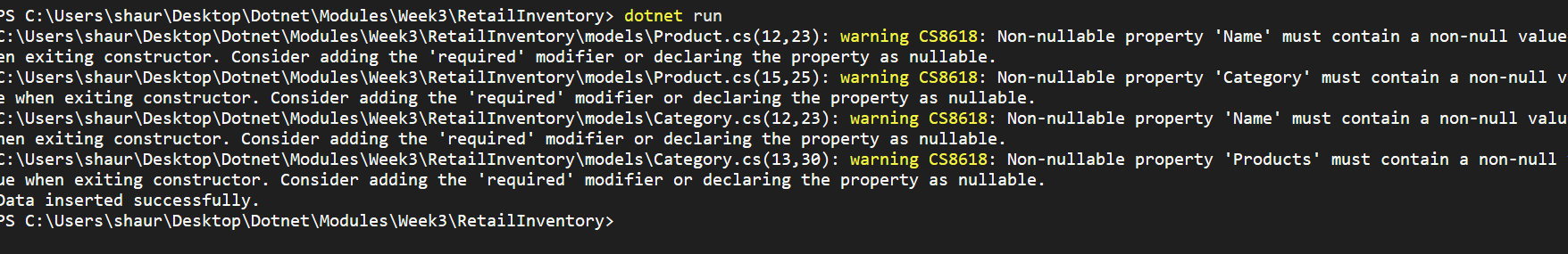
Here we can see a database that has been made and is named ***RetailDb.***

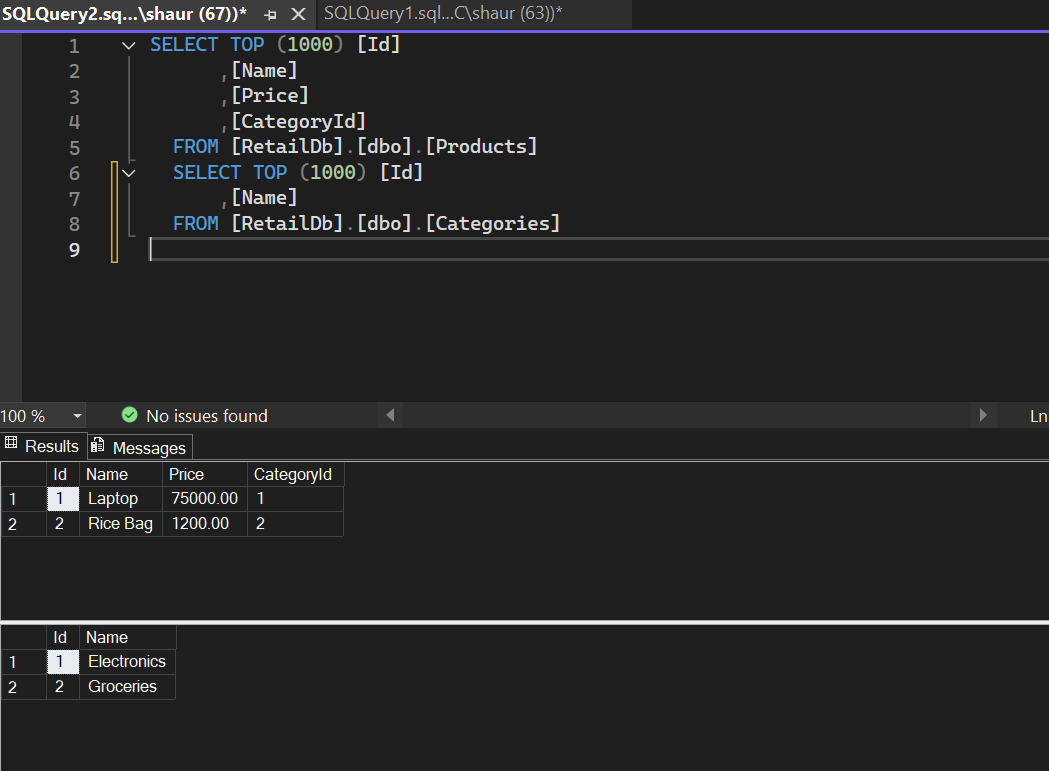
## Lab – 4: Inserting Initial Data into the Database

Code:

|  |
| --- |
| using RetailInventory.Database;  using RetailInventory.models;  using System;  using System.Threading.Tasks;    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("Data inserted successfully.");  }  } |

Output:





## Lab –5: Retrieving Data from Database

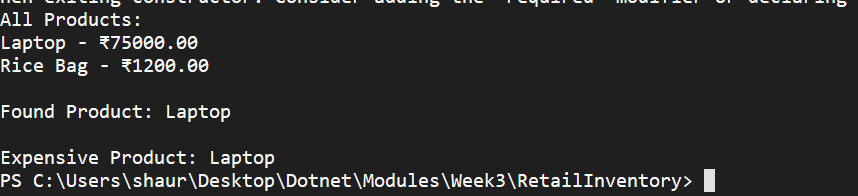
*Code (RetailInventoryServices.cs):*

|  |
| --- |
| using Microsoft.EntityFrameworkCore;  using RetailInventory.Database;  using RetailInventory.models;  using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;    namespace RetailInventory.Services  {  public class RetailInventoryServices  {  private readonly AppDbContext \_appDbContext;    public RetailInventoryServices(AppDbContext appDbContext)  {  \_appDbContext = appDbContext;  }    public async Task<List<Product>> GetAllProducts()  {  return await \_appDbContext.Products.ToListAsync();  }    public async Task<Product?> FindProductByIdAsync(int id)  {  return await \_appDbContext.Products.FindAsync(id);  }    public async Task<Product?> FindFirstProductWithPriceGreaterThanAsync(decimal price)  {  return await \_appDbContext.Products.FirstOrDefaultAsync(p => p.Price > price);  }  }  } |

*Code (Program.cs):*

|  |
| --- |
| using RetailInventory.Database;  using RetailInventory.models;  using RetailInventory.Services;  using System;  using System.Threading.Tasks;    class Program  {  static async Task Main(string[] args)  {  var dbContext = new AppDbContext();    var inventoryService = new RetailInventoryServices(dbContext);    var allProducts = await inventoryService.GetAllProducts();  Console.WriteLine("All Products:");  foreach (var product in allProducts)  {  Console.WriteLine($"{product.Name} - ₹{product.Price}");  }    var foundProduct = await inventoryService.FindProductByIdAsync(1);  Console.WriteLine($"\nFound Product: {foundProduct?.Name ?? "Not Found"}");    var expensiveProducts = await inventoryService.FindFirstProductWithPriceGreaterThanAsync(50000);  Console.WriteLine($"\nExpensive Product: {expensiveProducts?.Name ?? "None found"}");  }  } |

*Output:*



## Lab – 6: Updating and Deleting the Records

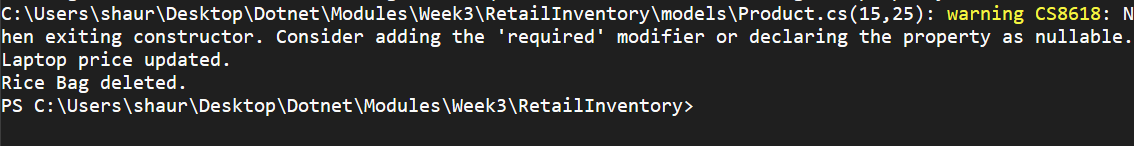
*Code (RetailInventoryServices.cs):*

|  |
| --- |
| public async Task<bool> UpdateProductPriceAsync(string productName, decimal newPrice)  {  var product = await \_appDbContext.Products.FirstOrDefaultAsync(p => p.Name == productName);  if (product != null)  {  product.Price = newPrice;  await \_appDbContext.SaveChangesAsync();  return true;  }  return false;  }    public async Task<bool> DeleteProductByNameAsync(string productName)  {  var product = await \_appDbContext.Products.FirstOrDefaultAsync(p => p.Name == productName);  if (product != null)  {  \_appDbContext.Products.Remove(product);  await \_appDbContext.SaveChangesAsync();  return true;  }  return false;  } |

*Code (Program.cs):*

|  |
| --- |
| using RetailInventory.Database;  using RetailInventory.models;  using RetailInventory.Services;  using System;  using System.Threading.Tasks;    class Program  {  static async Task Main(string[] args)  {  var dbContext = new AppDbContext();    var inventoryService = new RetailInventoryServices(dbContext);    bool updated = await inventoryService.UpdateProductPriceAsync("Laptop", 70000);  Console.WriteLine(updated ? "Laptop price updated." : "Laptop not found.");    bool deleted = await inventoryService.DeleteProductByNameAsync("Rice Bag");  Console.WriteLine(deleted ? "Rice Bag deleted." : "Rice Bag not found.");    }  } |

Output:



## Lab – 7: Writing Queries with LINQ

*Code (ProductDto.cs):*

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;    namespace RetailInventory.DTO  {  public class ProductDto  {  public string Name { get; set; }  public decimal Price { get; set; }  }  } |

*Code (RetailInventoryServices.cs):*

|  |
| --- |
| public async Task<List<Product>> GetFilteredSortedProductsAsync(decimal minPrice)  {  return await \_appDbContext.Products  .Where(p => p.Price > minPrice)  .OrderByDescending(p => p.Price)  .ToListAsync();  }    public async Task<List<ProductDto>> GetProductDTOsAsync()  {  return await \_appDbContext.Products  .Select(p => new ProductDto  {  Name = p.Name,  Price = p.Price  })  .ToListAsync();  } |

*Code (Program.cs):*

|  |
| --- |
| *using RetailInventory.Database;*  *using RetailInventory.models;*  *using RetailInventory.Services;*  *using System;*  *using System.Threading.Tasks;*    *class Program*  *{*  *static async Task Main(string[] args)*  *{*  *var dbContext = new AppDbContext();*    *var inventoryService = new RetailInventoryServices(dbContext);*    *var filteredProducts = await inventoryService.GetFilteredSortedProductsAsync(1000);*  *Console.WriteLine("\nFiltered Products (> ₹1000):");*  *foreach (var p in filteredProducts)*  *Console.WriteLine($"{p.Name} - ₹{p.Price}");*    *var productDTOs = await inventoryService.GetProductDTOsAsync();*  *Console.WriteLine("\nProduct DTOs:");*  *foreach (var dto in productDTOs)*  *Console.WriteLine($"{dto.Name} - ₹{dto.Price}");*  *}*  *}* |

*Output:*

