EF Core 8.0 Guided Hands-On Exercises

Lab 1: Understanding ORM with a Retail Inventory System

Create Console App**:**

dotnet new console -n RetailInventory

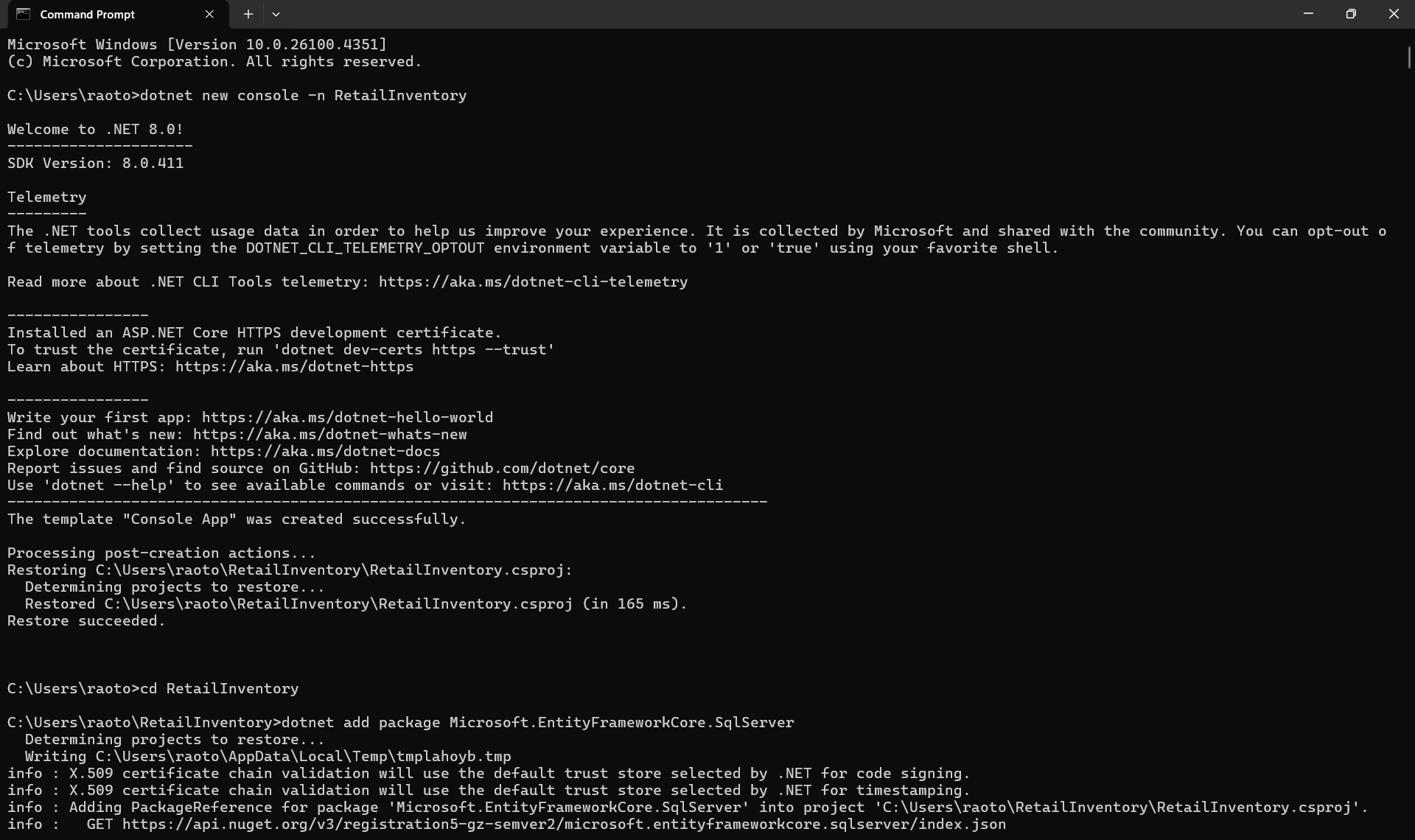
cd RetailInventory

Install EF Core Packages:

dotnet add package Microsoft.EntityFrameworkCore.SqlServer

dotnet add package Microsoft.EntityFrameworkCore.Design

**OUTPUT:**



Lab 2: Setting Up the Database Context

public class Category {

public int Id { get; set; }

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

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

}

public class Product {

public int Id { get; set; }

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

public decimal Price { get; set; }

public int CategoryId { get; set; }

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

}

public class AppDbContext : DbContext {

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

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

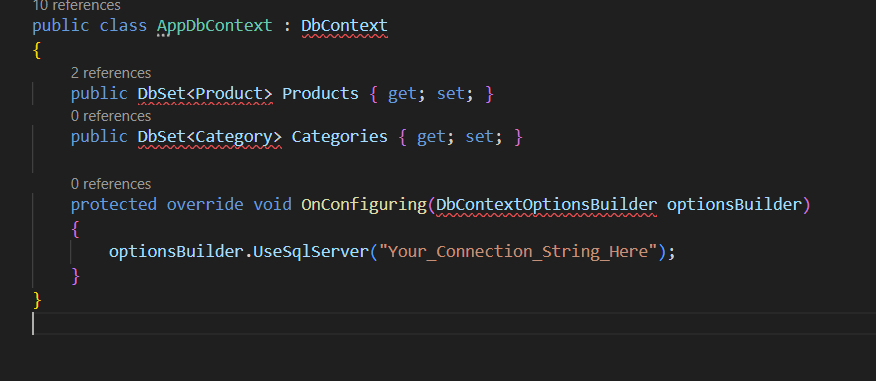
protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder) {

optionsBuilder.UseSqlServer("Your\_Connection\_String\_Here");

}

}

**OUTPUT:**



Lab 3: Creating and Applying Migrations

# Install EF Core CLI

dotnet tool install --global dotnet-ef

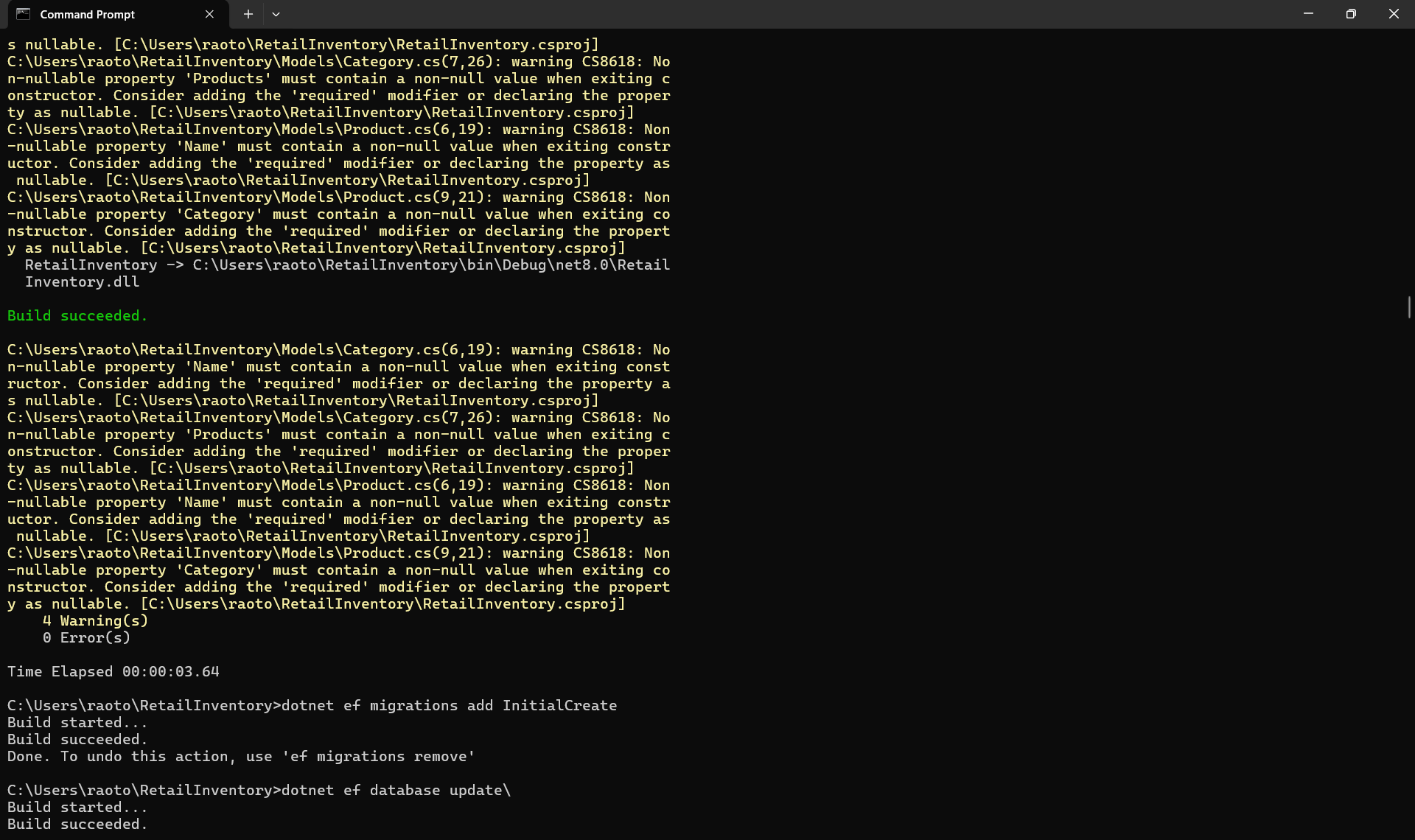
# Add migration

dotnet ef migrations add InitialCreate

# Apply migration

dotnet ef database update

**OUTPUT:**



Lab 4: Inserting Initial Data

using var context = new AppDbContext();

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

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

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

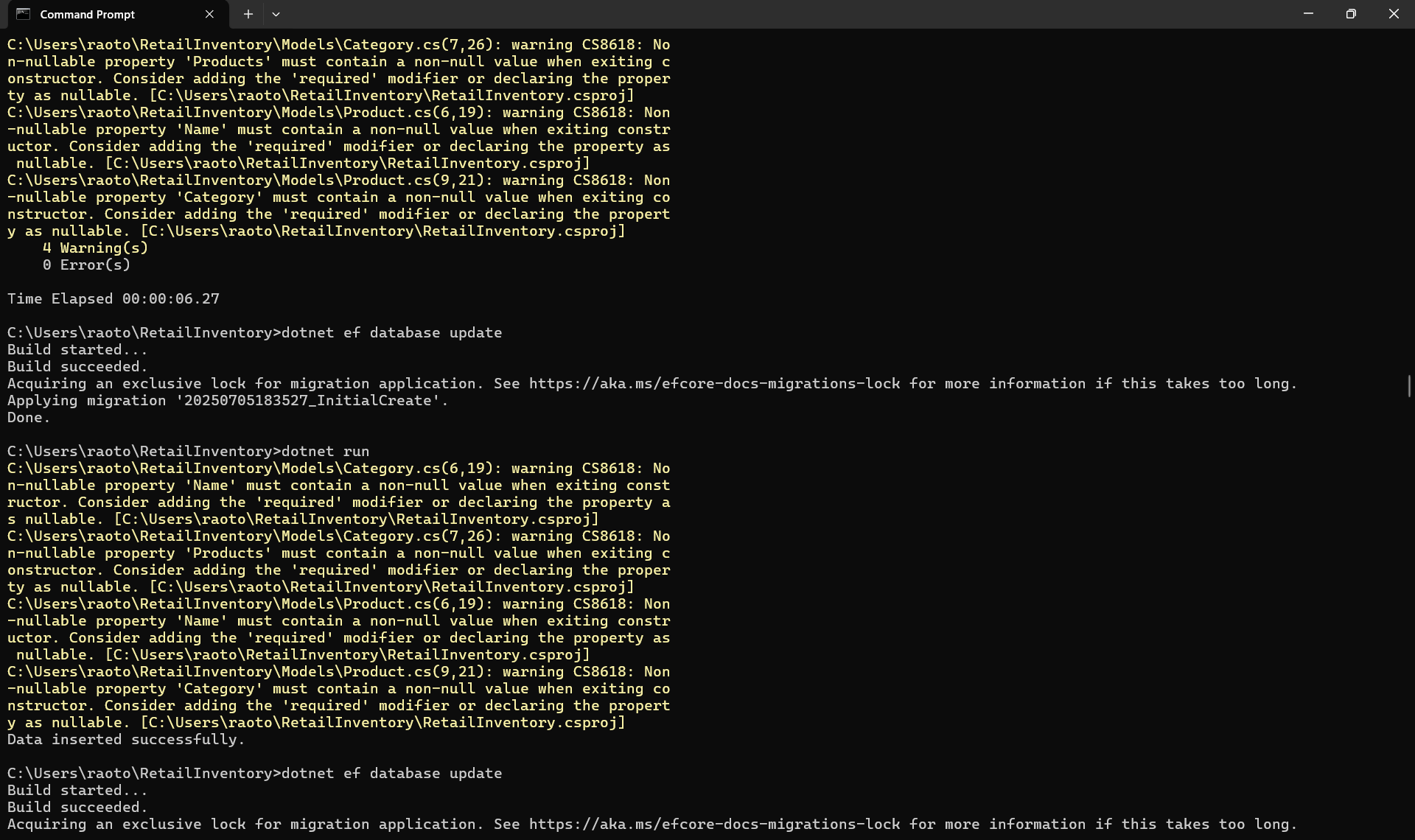
var product1 = new Product { Name = "Laptop", Price = 75000, Category = electronics };

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

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

await context.SaveChangesAsync();

**OUTPUT:**



Lab 5: Retrieving Data

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

foreach (var p in products)

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

**Lab 6: Updating and Deleting Records**

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

if (product != null) {

product.Price = 70000;

await context.SaveChangesAsync();

}

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

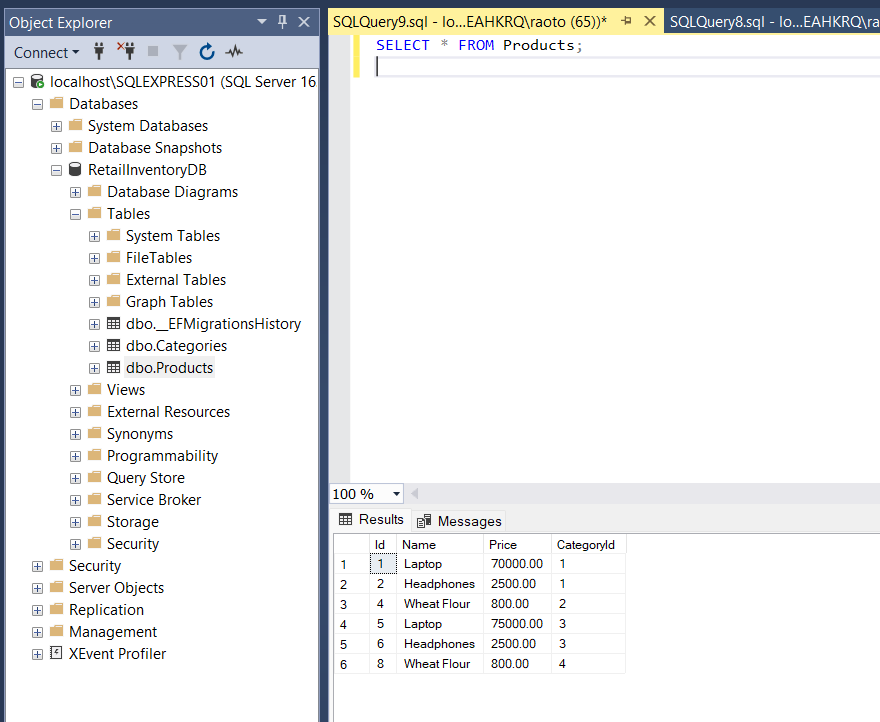
if (toDelete != null) {

context.Products.Remove(toDelete);

await context.SaveChangesAsync();

}

**OUTPUT:**



### Lab 7: Writing Queries with LINQ

var filtered = await context.Products

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

.OrderByDescending(p => p.Price)

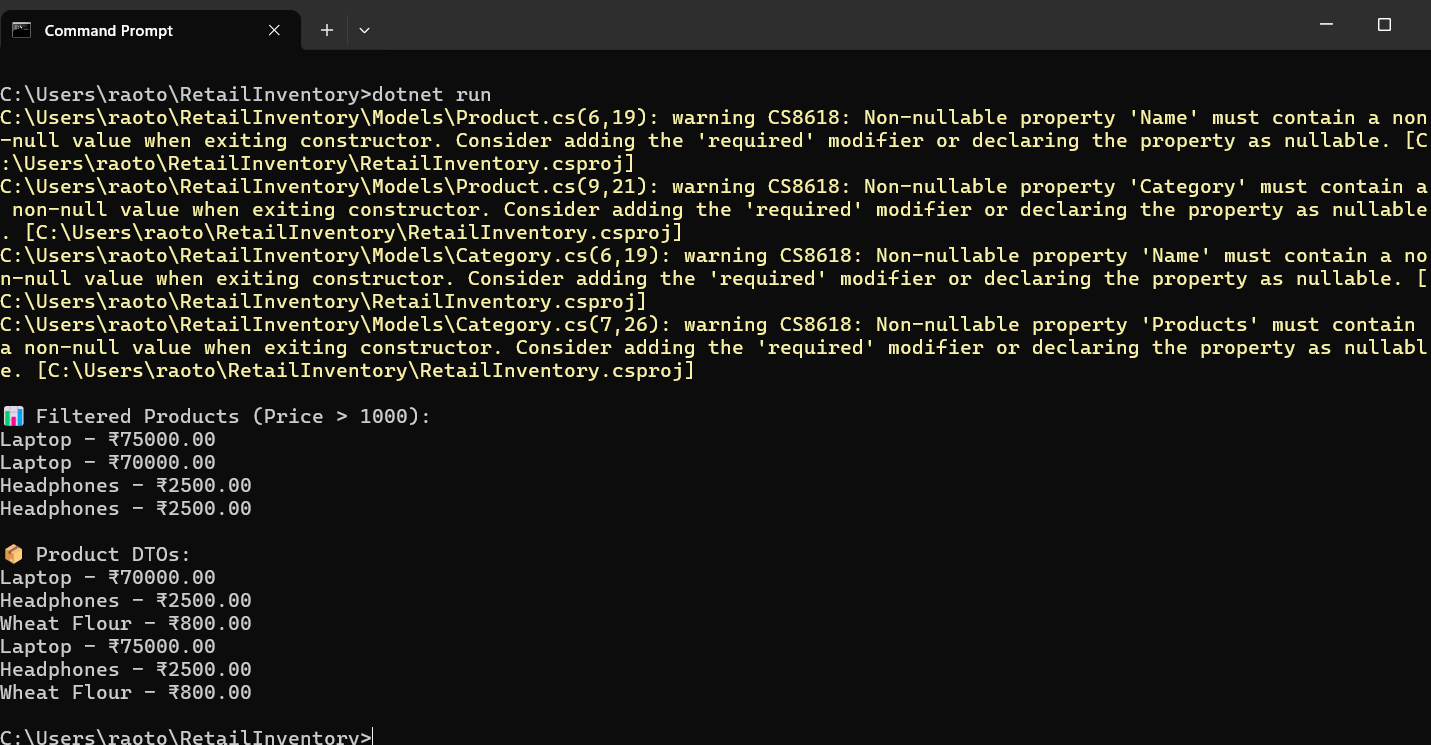
.ToListAsync();

var productDTOs = await context.Products

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

.ToListAsync();

**OUTPUT:**



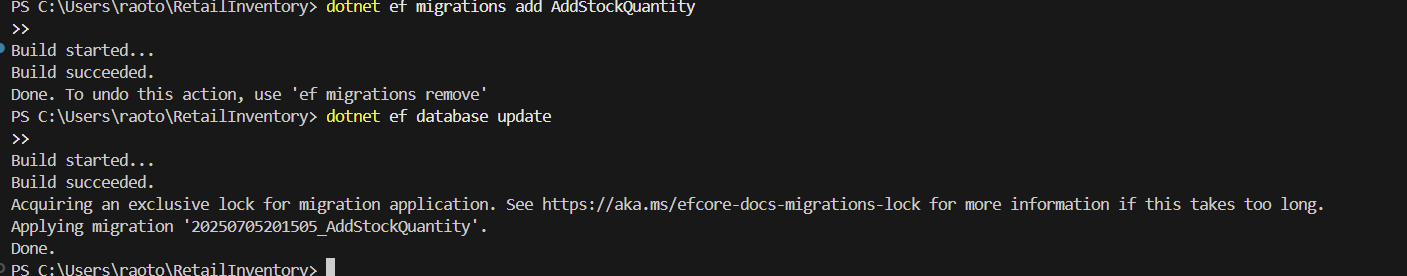
### Lab 8:Managing Migrations - Adding StockQuantity

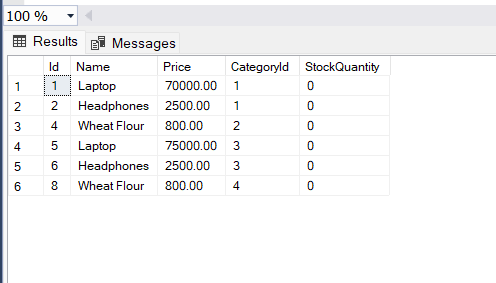
public int StockQuantity { get; set; }

dotnet ef migrations add AddStockQuantity

dotnet ef database update

**OUTPUT:**





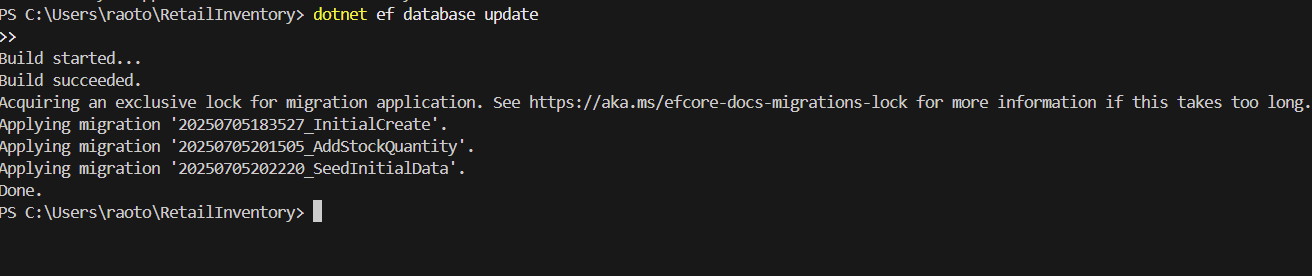
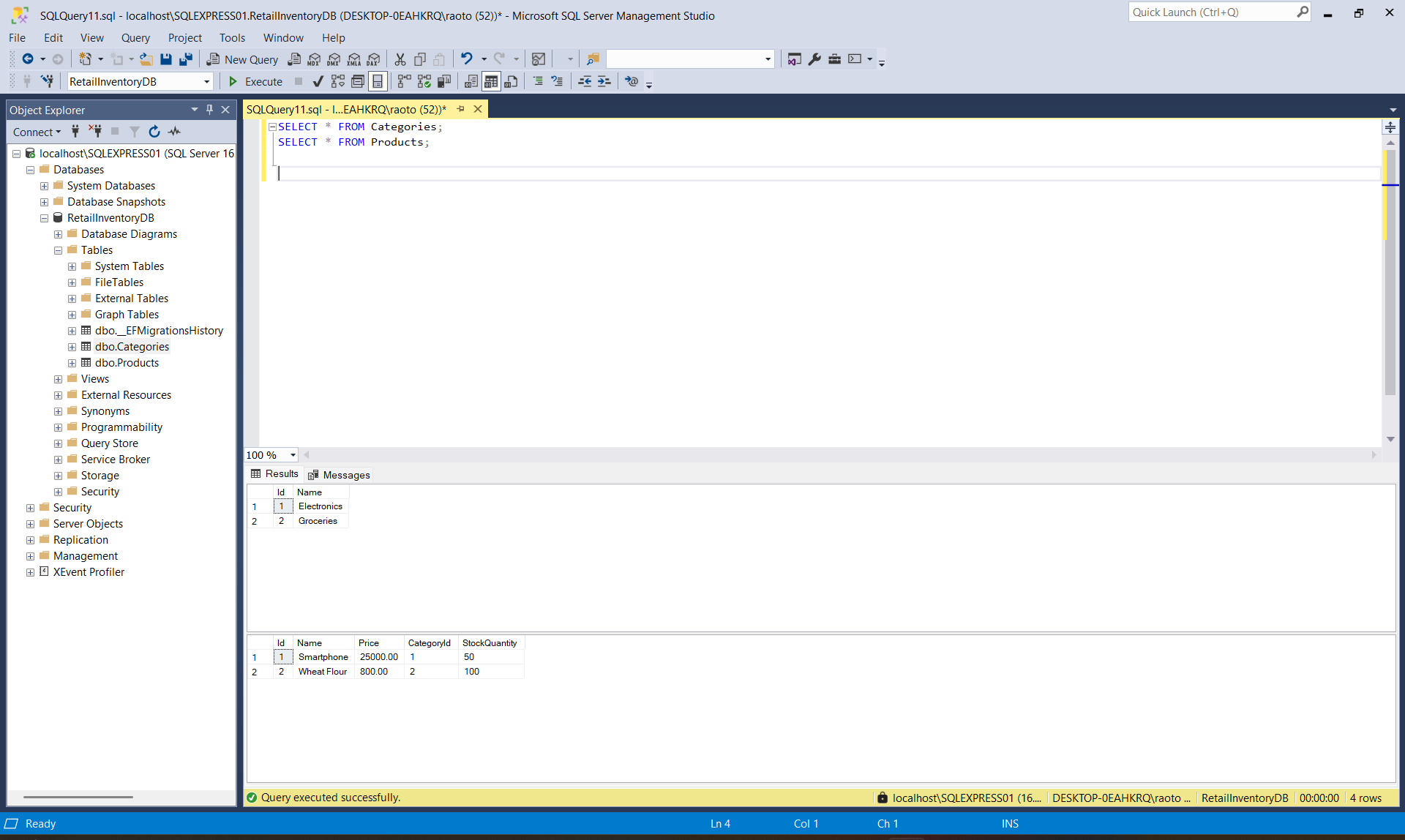
### Lab 9: Seeding Data During Migrations

### modelBuilder.Entity<Category>().HasData(

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

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

**);**

**OUTPUT:**

### modelBuilder.Entity<Product>().HasData(

### new Product { Id = 1, Name = "Smartphone", Price = 25000, CategoryId = 1, StockQuantity = 50 },

### new Product { Id = 2, Name = "Wheat Flour", Price = 800, CategoryId = 2, StockQuantity = 100 }

### );

### Lab 10: Loading Strategies

var products = await context.Products.Include(p => p.Category).ToListAsync();

var product = await context.Products.FirstAsync();

await context.Entry(product).Reference(p => p.Category).LoadAsync();

**OUTPUT:**



### Lab 11: One-to-One and Many-to-Many Relationships

### Step 1: One-to-One Relationship

### public class ProductDetail

### {

### public int ProductDetailId { get; set; }

### public string WarrantyInfo { get; set; }

### public int ProductId { get; set; }

### public Product Product { get; set; }

### }

### modelBuilder.Entity<Product>()

### .HasOne(p => p.ProductDetail)

### .WithOne(pd => pd.Product)

### .HasForeignKey<ProductDetail>(pd => pd.ProductId);

**OUTPUT:**

### C:\week3\lab 11.png

**Step 2: Many-to-Many Relationship**

public class Tag

{

public int Id { get; set; }

public string Name { get; set; }

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

}

public class Product

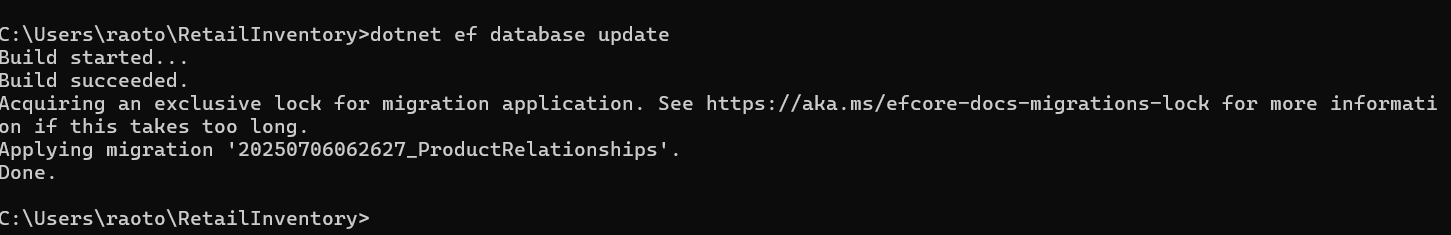
{

...

public List<Tag> Tags { get; set; }

}

**OUTPUT:**



**Lab 12: Handling Circular References**

public class ProductDTO {

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

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

}

var productDTOs = await context.Products

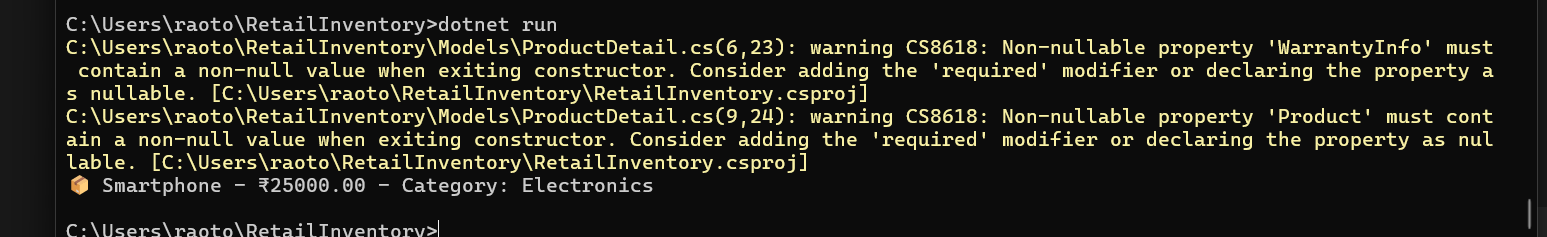
.Select(p => new ProductDTO {

Name = p.Name,

CategoryName = p.Category.Name

}).ToListAsync();

**OUTPUT:**

****

**Lab 13: Query Caching and Tracking**

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

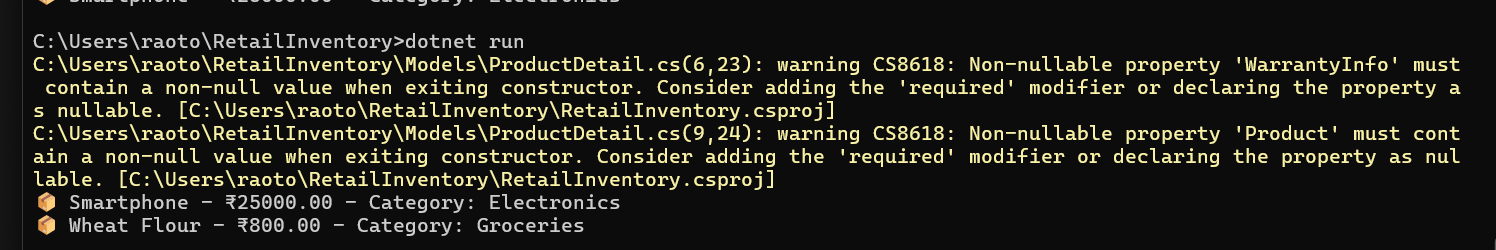
static readonly Func<AppDbContext, decimal, Task<List<Product>>> \_expensiveProducts =

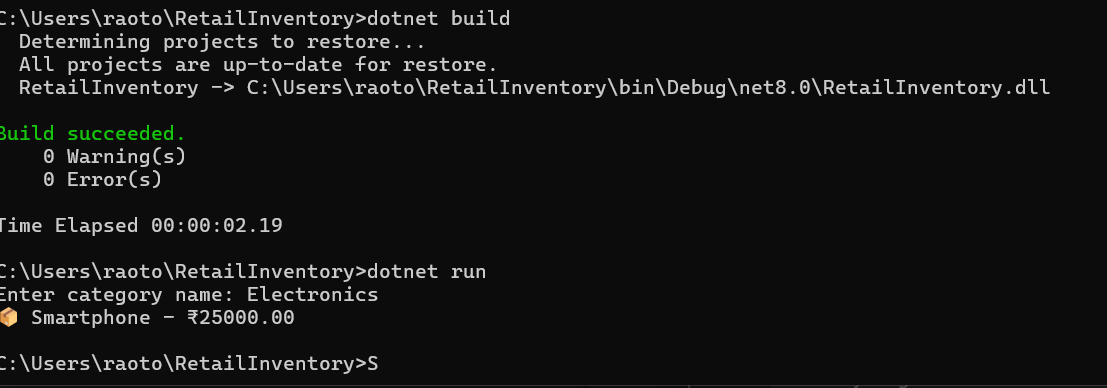
EF.CompileAsyncQuery((AppDbContext ctx, decimal price) =>

ctx.Products.Where(p => p.Price > price));

var result = await \_expensiveProducts(context, 10000);

**OUTPUT:**





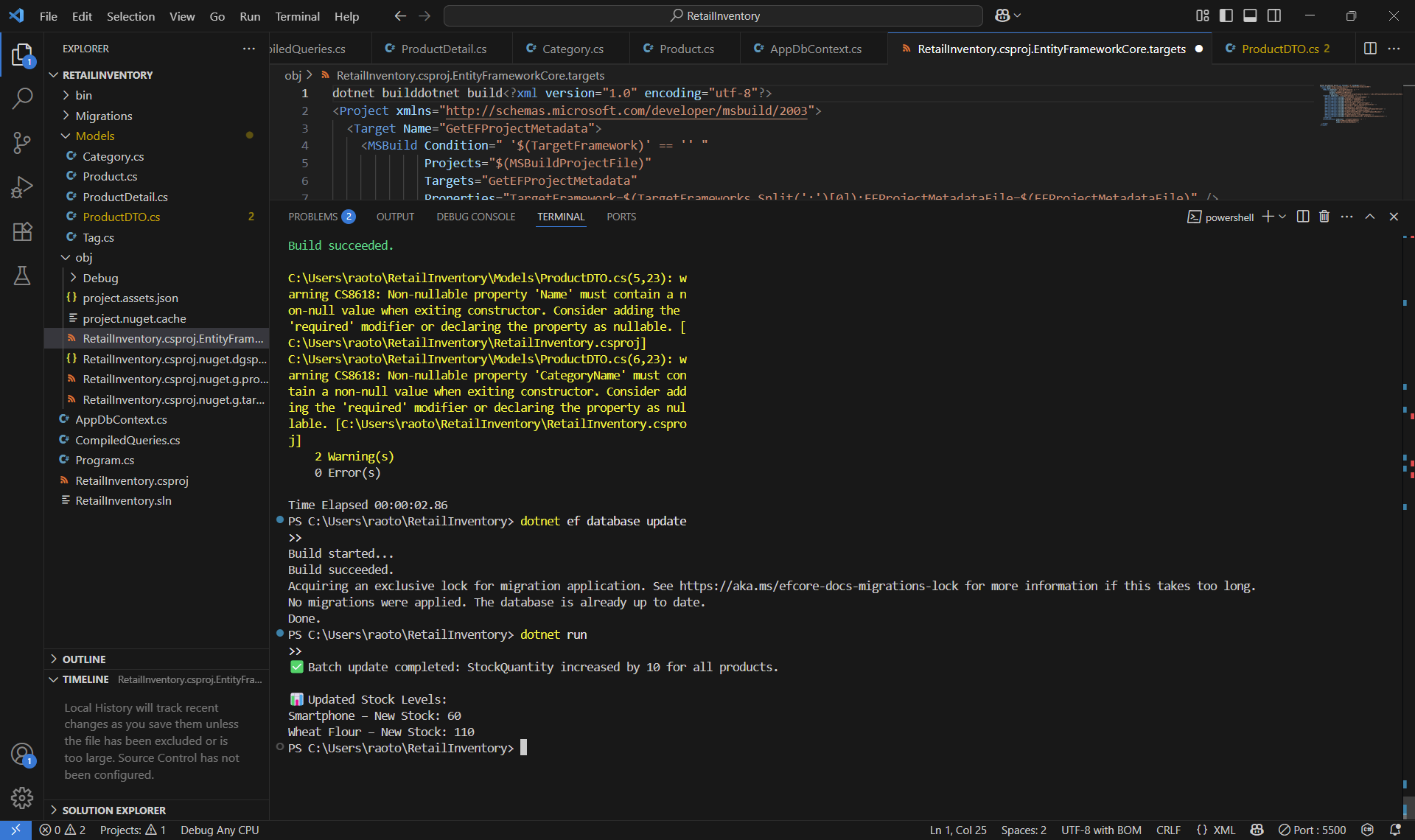
**Lab 14: Bulk Operations**

dotnet add package EFCore.BulkExtensions

products.ForEach(p => p.StockQuantity += 10);

await context.BulkUpdateAsync(products);

**OUTPUT:**



**Lab 15: Concurrency Handling with RowVersion**

**[Timestamp]**

**public byte[] RowVersion { get; set; }**

**try**

**{**

**await context.SaveChangesAsync();**

**}**

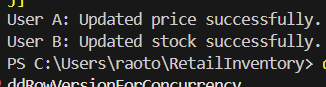
**catch (DbUpdateConcurrencyException)**

**{**

**Console.WriteLine("Concurrency conflict detected.");**

**}**

**OUTPUT:**

****