**EF Core 8.0 Guided Additional\_Hands\_On Exercises**

**Submitted By:** Gyarina Rauniyar  
**Lab 6: Updating and Deleting Records using EF Core**

### **Scenario:**

The store updates product prices and removes discontinued items.

### **Objective:**

To update and delete records using Entity Framework Core.

## **Steps and Code Implementation:**

### **Step 1: Setup**

Ensured the following namespaces are added at the top of Program.cs:

using System;  
using System.Threading.Tasks;  
using RetailStoreContextLab;  
using Microsoft.EntityFrameworkCore;

### **Step 2: Main Method Logic for Updating and Deleting Records**

The Main method was updated as follows:

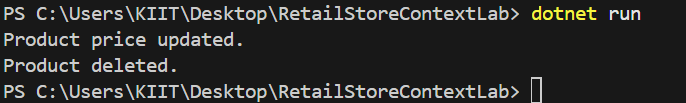
class Program  
{  
 static async Task Main(string[] args)  
 {  
 using var context = new AppDbContext();  
  
 // Update a Product  
 var product = await context.Products.FirstOrDefaultAsync(p => p.Name == "Laptop");  
 if (product != null)  
 {  
 product.Price = 70000;  
 await context.SaveChangesAsync();  
 Console.WriteLine("Product price updated.");  
 }  
 else  
 {  
 Console.WriteLine("Product not found.");  
 }  
  
 // Delete a Product  
 var toDelete = await context.Products.FirstOrDefaultAsync(p => p.Name == "Rice Bag");  
 if (toDelete != null)  
 {  
 context.Products.Remove(toDelete);  
 await context.SaveChangesAsync();  
 Console.WriteLine("Product deleted.");  
 }  
 else  
 {  
 Console.WriteLine("Product to delete not found.");  
 }  
 }  
}

## **Step 3: Built and Ran the Application**

Executed the following commands in the terminal:

dotnet build  
dotnet run

### **OutputScreenshot:**



## **Conclusion:**

In this lab, I have successfully learned how to:

* Update an existing product’s price in the database.
* Delete a product from the database.

**Lab 7: Writing Queries with LINQ using EF Core**

### **Scenario:**

The store wants to filter and sort products for reporting.

### **Objective:**

To use LINQ methods like Where, Select, OrderBy, and project data into DTOs using Entity Framework Core.

## **Steps and Code Implementation:**

### **Step 1: Setup**

Ensured the following namespaces are added at the top of Program.cs:

using System;  
using System.Threading.Tasks;  
using RetailStoreContextLab;  
using Microsoft.EntityFrameworkCore;  
using System.Linq;

### **Step 2: Main Method Logic for Filtering, Sorting, and DTO Projection**

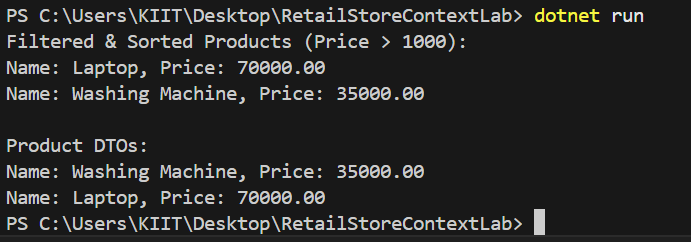
class Program  
{  
 static async Task Main(string[] args)  
 {  
 using var context = new AppDbContext();  
  
 // Filter and Sort Products  
 var filtered = await context.Products  
 .Where(p => p.Price > 1000)  
 .OrderByDescending(p => p.Price)  
 .ToListAsync();  
  
 Console.WriteLine("Filtered & Sorted Products (Price > 1000):");  
 foreach (var product in filtered)  
 {  
 Console.WriteLine($"Name: {product.Name}, Price: {product.Price:F2}");  
 }  
  
 // Project into DTOs  
 var productDTOs = await context.Products  
 .Select(p => new { p.Name, p.Price })  
 .ToListAsync();  
  
 Console.WriteLine("\nProduct DTOs:");  
 foreach (var dto in productDTOs)  
 {  
 Console.WriteLine($"Name: {dto.Name}, Price: {dto.Price:F2}");  
 }  
 }  
}

## **Step 3: Built and Ran the Application**

Executed the following commands in the terminal:

dotnet build  
dotnet run

### **Output Screenshot:**

****

## **Note:**

(We might notice that the product **“Rice Bag”** was missing from the output, even though its price is 1200 and satisfies the filter condition. That was because during **Lab 6**, the “Rice Bag” product was deleted from the database. As a result, it no longer appeared in the results for Lab 7.)

## **Conclusion:**

In this lab, I have successfully learned how to:

* Use LINQ to filter and sort records from the database.
* Project selected fields into anonymous DTOs for reporting.
* Understand how data modifications from previous operations (like deletions) affect query results.

Entity Framework Core and LINQ provide powerful and expressive tools to perform complex data queries with minimal code.