**LAB 6:EF CORE 8.0 HOL**

**Aim:** Updating and Deleting Records

**Scenario:** The store updates product prices and removes discontinued items.

**Models\Product.cs:**

namespace RetailInventoryApp.Models;

public class Product

{

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

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

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

}

**Models\Category.cs:**

namespace RetailInventoryApp.Models;

public class Category

{

public int CategoryId { get; set; }

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

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

}

**Models\Stock.cs:**

namespace RetailInventoryApp.Models;

public class Stock

{

public int StockId { get; set; }

public int QuantityAvailable { get; set; }

public DateTime LastChecked { get; set; }

public int ProductId { get; set; }

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

}

**Models\Suppliers:**

namespace RetailInventoryApp.Models;

public class Supplier

{

public int SupplierId { get; set; }

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

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

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

}

**Data\RetailContext.cs:**

using Microsoft.EntityFrameworkCore;

using RetailInventoryApp.Models;

namespace RetailInventoryApp.Data;

public class RetailContext : DbContext

{

public DbSet<Product> Products => Set<Product>();

public DbSet<Category> Categories => Set<Category>();

public DbSet<Supplier> Suppliers => Set<Supplier>();

public DbSet<Stock> Stocks => Set<Stock>();

protected override void OnConfiguring(DbContextOptionsBuilder options)

=> options.UseSqlite("Data Source=retail.db");

}

**Program.cs:**

using RetailInventoryApp.Data;

using RetailInventoryApp.Models;

using Microsoft.EntityFrameworkCore;

using var context = new RetailContext();

await context.Database.EnsureCreatedAsync();

if (!await context.Products.AnyAsync())

{

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

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

var supplier1 = new Supplier { Name = "Tech Supplier", Email = "tech@supplier.com" };

var supplier2 = new Supplier { Name = "Balaji Suppliers", Email = "balaji@supplier.com" };

var laptop = new Product

{

Name = "Laptop",

Price = 75000,

Category = electronics,

Supplier = supplier1,

Stock = new Stock { QuantityAvailable = 20, LastChecked = DateTime.Now }

};

var riceBag = new Product

{

Name = "Rice Bag",

Price = 1200,

Category = groceries,

Supplier = supplier2,

Stock = new Stock { QuantityAvailable = 50, LastChecked = DateTime.Now }

};

await context.AddRangeAsync(electronics, groceries, supplier1, supplier2, laptop, riceBag);

await context.SaveChangesAsync();

Console.WriteLine(" Initial data inserted.");

}

async Task DisplayProducts(string title)

{

Console.WriteLine($"\n {title}");

var products = await context.Products

.Include(p => p.Category)

.Include(p => p.Supplier)

.Include(p => p.Stock)

.ToListAsync();

if (products.Any())

{

foreach (var p in products)

{

Console.WriteLine($"{p.ProductId}: {p.Name} - ₹{p.Price} ({p.Category.Name}) | Qty: {p.Stock.QuantityAvailable} | Supplier: {p.Supplier.Name}");

}

}

else

{

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

}

}

bool exit = false;

while (!exit)

{

await DisplayProducts("Current Products");

Console.WriteLine("\nChoose an action:\n1. Update Product\n2. Delete Product\n3. Exit");

Console.Write("Enter your choice: ");

string? choice = Console.ReadLine();

switch (choice)

{

case "1":

Console.Write("Enter Product ID to update: ");

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

{

var product = await context.Products

.Include(p => p.Stock)

.Include(p => p.Supplier)

.FirstOrDefaultAsync(p => p.ProductId == updateId);

if (product != null)

{

Console.Write($"Current Price: ₹{product.Price}. Enter new price: ");

if (decimal.TryParse(Console.ReadLine(), out decimal newPrice))

product.Price = newPrice;

Console.Write($"Current Quantity: {product.Stock.QuantityAvailable}. Enter new quantity: ");

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

product.Stock.QuantityAvailable = newQty;

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

Console.Write("Change supplier to (1: Tech Supplier, 2: Balaji Suppliers): ");

var supplierChoice = Console.ReadLine();

var newSupplier = await context.Suppliers.FirstOrDefaultAsync(s =>

(supplierChoice == "1" && s.Name == "Tech Supplier") ||

(supplierChoice == "2" && s.Name == "Balaji Suppliers"));

if (newSupplier != null)

product.Supplier = newSupplier;

product.Stock.LastChecked = DateTime.Now;

await context.SaveChangesAsync();

Console.WriteLine(" Product updated!");

}

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

}

break;

case "2":

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

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

{

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

if (product != null)

{

context.Products.Remove(product);

await context.SaveChangesAsync();

Console.WriteLine("🗑 Product deleted.");

}

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

}

break;

case "3":

exit = true;

Console.WriteLine(" Exiting...");

break;

default:

Console.WriteLine(" Invalid option.");

break;

}

if (!exit)

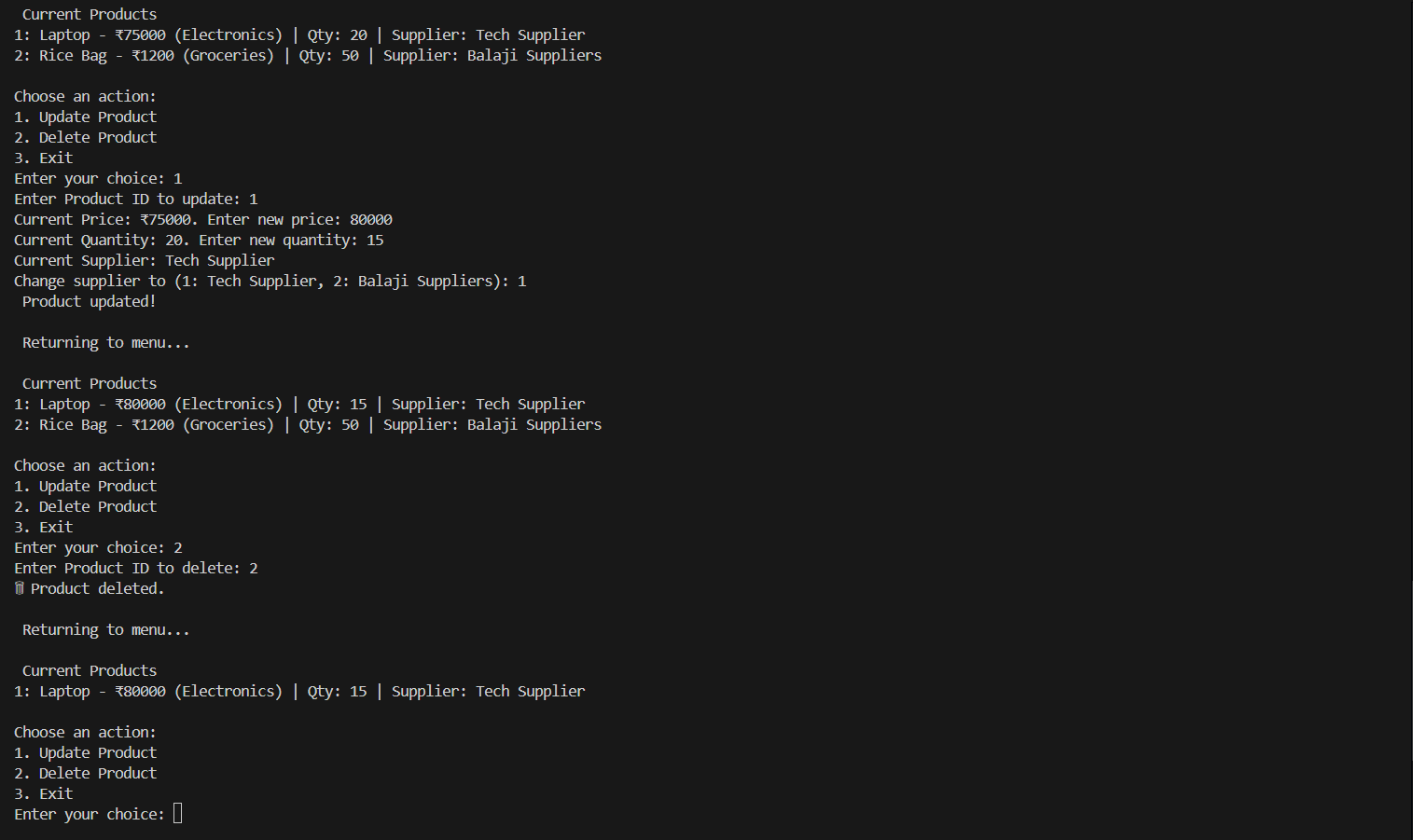
{

Console.WriteLine("\n Returning to menu...");

}

}

**Output:**

****