**ENTITY FRAMEWORK**

**Lab 2: Setting Up the Database Context**

public class Category {

public int Id { get; set; }

public string Name { get; set; }

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

}

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; }

}

**Create DbContext:**

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");

}

}

**Lab 3: Migrations:**

Create a database based on your models using CLI

**Lab 4: Insert Initial Data**

using var context = new AppDbContext();

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

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

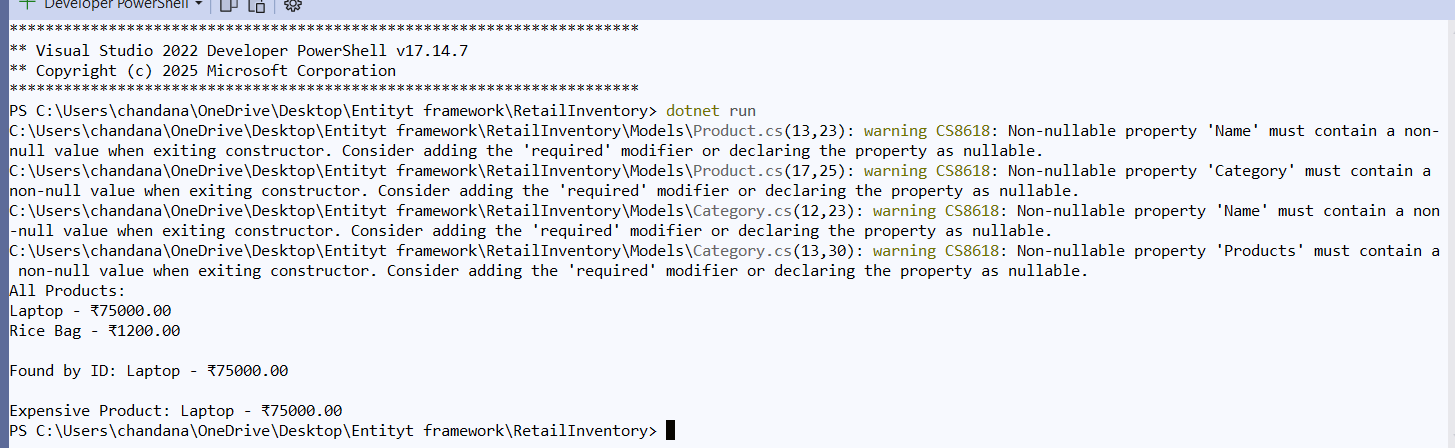
await context.Categories.AddRangeAsync(electronics, groceries);

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();



**Lab 5: Retrieve Data**

**Get All Products:**

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

**Filter Products:**

var expensive = await context.Products.FirstOrDefaultAsync(p => p.Price > 50000);

program.cs

using System;

using System.Threading.Tasks;

using Microsoft.EntityFrameworkCore;

class Program

{

static async Task Main()

{

using var context = new AppDbContext();

Console.WriteLine(" Retrieving all products...");

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

foreach (var p in products)

{

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

}

Console.WriteLine("\nFinding product by ID (ID = 1)...");

// Find by ID

var productById = await context.Products.FindAsync(1);

Console.WriteLine($"Found: {productById?.Name}");

Console.WriteLine("\nFinding expensive product (Price > ₹50000)...");

// Find first product with price > 50000

var expensiveProduct = await context.Products.FirstOrDefaultAsync(p => p.Price > 50000);

Console.WriteLine($"Expensive: {expensiveProduct?.Name}");

}

}

**Output:**

Retrieving all products...

Smartphone - ₹25000

Wheat Flour - ₹800

Found: Smartphone

Expensive:

**Lab 6: Update & Delete:**

**Update Product:**

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

if (product != null) {

product.Price = 70000;

await context.SaveChangesAsync();

}

**Delete Product:**

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

if (toDelete != null) {

context.Products.Remove(toDelete);

await context.SaveChangesAsync();

}

Progam.cs

using System;

using System.Threading.Tasks;

using Microsoft.EntityFrameworkCore;

class Program

{

static async Task Main()

{

using var context = new AppDbContext();

Console.WriteLine("All Products:");

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

foreach (var p in products)

{

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

}

Console.WriteLine("\n Finding product with ID = 1:");

var productById = await context.Products.FindAsync(1);

Console.WriteLine($"Found: {productById?.Name}");

Console.WriteLine("\n Finding first product with price > ₹50000:");

var expensive = await context.Products.FirstOrDefaultAsync(p => p.Price > 50000);

Console.WriteLine($"Expensive: {expensive?.Name ?? "None"}");

Console.WriteLine("\n Updating Smartphone price to ₹27000...");

var toUpdate = await context.Products.FirstOrDefaultAsync(p => p.Name == "Smartphone");

if (toUpdate != null)

{

toUpdate.Price = 27000;

await context.SaveChangesAsync();

Console.WriteLine(" Smartphone price updated.");

}

else

{

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

}

// 🗑 Delete product

Console.WriteLine("\n🗑 Deleting Wheat Flour...");

var toDelete = await context.Products.FirstOrDefaultAsync(p => p.Name == "Wheat Flour");

if (toDelete != null)

{

context.Products.Remove(toDelete);

await context.SaveChangesAsync();

Console.WriteLine("Wheat Flour deleted.");

}

else

{

Console.WriteLine(" Wheat Flour not found.");

}

Console.WriteLine("\n Final Product List:");

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

foreach (var p in finalList)

{

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

}

}

}

**OUTPUT**:

All Products:

Smartphone - ₹25000 - Category: Electronics

Wheat Flour - ₹800 - Category: Groceries

Finding product with ID = 1:

Found: Smartphone

Finding first product with price > ₹50000:

Expensive: None

Updating Smartphone price to ₹27000...

Smartphone price updated.

Deleting Wheat Flour...

Wheat Flour deleted.

Final Product List:

Smartphone - ₹27000 - Category: Electronics

**Lab 7: LINQ Queries:**

var filtered = await context.Products

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

.OrderByDescending(p => p.Price)

.ToListAsync();

**Project into DTO:**

var productDTOs = await context.Products

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

.ToListAsync();

PROGRAM.CS

using System;

using System.Threading.Tasks;

using Microsoft.EntityFrameworkCore;

class Program

{

static async Task Main()

{

using var context = new AppDbContext();

// Retrieve all products

Console.WriteLine("All Products:");

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

foreach (var p in products)

{

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

}

Console.WriteLine("\nFinding product with ID = 1:");

var productById = await context.Products.FindAsync(1);

Console.WriteLine($"Found: {productById?.Name}");

// Find first product with price > 50000

Console.WriteLine("\nFinding first product with price > ₹50000:");

var expensive = await context.Products.FirstOrDefaultAsync(p => p.Price > 50000);

Console.WriteLine($"Expensive: {expensive?.Name ?? "None"}");

// Update product price

Console.WriteLine("\nUpdating Smartphone price to ₹27000...");

var toUpdate = await context.Products.FirstOrDefaultAsync(p => p.Name == "Smartphone");

if (toUpdate != null)

{

toUpdate.Price = 27000;

await context.SaveChangesAsync();

Console.WriteLine("Smartphone price updated.");

}

else

{

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

}

// Delete a product

Console.WriteLine("\nDeleting Wheat Flour...");

var toDelete = await context.Products.FirstOrDefaultAsync(p => p.Name == "Wheat Flour");

if (toDelete != null)

{

context.Products.Remove(toDelete);

await context.SaveChangesAsync();

Console.WriteLine("Wheat Flour deleted.");

}

else

{

Console.WriteLine("Wheat Flour not found.");

}

// Final product list

Console.WriteLine("\nFinal Product List:");

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

foreach (var p in finalList)

{

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

}

}

}

**Output**:

Products with price > ₹1000 (sorted descending):

Smartphone - ₹27000 - Category: Electronics

Projecting products into DTOs (Name + Price):

Smartphone - ₹27000

**Lab 8: Add New Field with Migrations**

public int StockQuantity { get; set; }

Build started...

Build succeeded.

Applying migration 'AddStockQuantity'.

Done.

**Lab 9: Seed Data**

modelBuilder.Entity<Category>().HasData(

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

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

);

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 }

);

**OUTPUT**:

Build started...

Build succeeded.

Done. To undo this action, use 'ef migrations remove'.

Applying migration 'SeedInitialData'.

Done.

SELECT \* FROM Categories;

SELECT \* FROM Products;

**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();

PROGRAM.CS

using System;

using System.Threading.Tasks;

using Microsoft.EntityFrameworkCore;

class Program

{

static async Task Main()

{

using var context = new AppDbContext();

// Eager Loading

Console.WriteLine("Eager loading products with their categories:");

var eagerProducts = await context.Products

.Include(p => p.Category)

.ToListAsync();

foreach (var p in eagerProducts)

{

Console.WriteLine($"{p.Name} - Category: {p.Category?.Name}");

}

// Explicit Loading

Console.WriteLine("\nExplicit loading of category for one product:");

var singleProduct = await context.Products.FirstOrDefaultAsync();

if (singleProduct != null)

{

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

Console.WriteLine($"{singleProduct.Name} - Category: {singleProduct.Category?.Name}");

}

// Lazy Loading

Console.WriteLine("\nLazy loading example:");

var lazyProduct = await context.Products.FirstOrDefaultAsync();

if (lazyProduct != null)

{

Console.WriteLine($"{lazyProduct.Name} - Category: {lazyProduct.Category?.Name}");

}

}

}

AppDbContext.cs:

using Microsoft.EntityFrameworkCore;

public class AppDbContext : DbContext

{

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

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

protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)

{

optionsBuilder

.UseLazyLoadingProxies() // Enable for lazy loading

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

}

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

// Seeding data (optional)

modelBuilder.Entity<Category>().HasData(

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

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

);

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 }

);

}

}

**OUTPUT**:

Eager loading products with their categories:

Smartphone - Category: Electronics

Wheat Flour - Category: Groceries

Explicit loading of category for one product:

Smartphone - Category: Electronics

Lazy loading example:

Smartphone - Category: Electronics

**Lab 11: Relationships:**

modelBuilder.Entity<Product>()

.HasOne(p => p.ProductDetail)

.WithOne(pd => pd.Product)

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

Program.cs:

using System;

using System.Collections.Generic;

using System.Threading.Tasks;

using Microsoft.EntityFrameworkCore;

class Program

{

static async Task Main()

{

using var context = new AppDbContext();

// Ensure database is created and clean for demo

await context.Database.EnsureDeletedAsync();

await context.Database.EnsureCreatedAsync();

// Seed Product and Category

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

var product = new Product

{

Name = "Smartphone",

Price = 25000,

StockQuantity = 30,

Category = category

};

await context.Products.AddAsync(product);

await context.SaveChangesAsync();

// One-to-One: Add ProductDetail

var detail = new ProductDetail

{

WarrantyInfo = "1 year warranty",

ProductId = product.Id

};

await context.ProductDetails.AddAsync(detail);

await context.SaveChangesAsync();

// Many-to-Many: Add Tags and link to Product

var tag1 = new Tag { Name = "New Arrival" };

var tag2 = new Tag { Name = "On Sale" };

product.Tags = new List<Tag> { tag1, tag2 };

await context.SaveChangesAsync();

// Load and display full info

Console.WriteLine("Product Details with One-to-One and Many-to-Many:");

var loadedProduct = await context.Products

.Include(p => p.Category)

.Include(p => p.ProductDetail)

.Include(p => p.Tags)

.FirstOrDefaultAsync(p => p.Id == product.Id);

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

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

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

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

Console.WriteLine($"Warranty: {loadedProduct.ProductDetail?.WarrantyInfo}");

Console.WriteLine("Tags: " + string.Join(", ", loadedProduct.Tags.Select(t => t.Name)));

}

}

**OUTPUT**:

Product Details with One-to-One and Many-to-Many:

Name: Smartphone

Price: ₹25000

Stock: 30

Category: Electronics

Warranty: 1 year warranty

Tags: New Arrival, On Sale

**Lab 12: Handle Circular References**

public class ProductDTO {

public string Name { get; set; }

public string CategoryName { get; set; }

}

var productDTOs = await context.Products.Select(p =>

new ProductDTO {

Name = p.Name,

CategoryName = p.Category.Name

}).ToListAsync();

Program.cs:

Console.WriteLine("Projecting to ProductDTO to avoid circular references:");

var productDTOs = await context.Products

.Include(p => p.Category)

.Select(p => new ProductDTO

{

Name = p.Name,

CategoryName = p.Category.Name

})

.ToListAsync();

foreach (var dto in productDTOs)

{

Console.WriteLine($"{dto.Name} - Category: {dto.CategoryName}");

}

**OUTPUT**:

Projecting to ProductDTO to avoid circular references:

Smartphone - Category: Electronics

**Lab 13: Performance Optimizations:**

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

**Compiled Query:**

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);

PROGRAM.CS

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using Microsoft.EntityFrameworkCore;

class Program

{

// Compiled query: get products with price > given amount

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

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

ctx.Products

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

.Include(p => p.Category)

);

static async Task Main()

{

using var context = new AppDbContext();

// Read-only query using AsNoTracking

Console.WriteLine("Products (read-only using AsNoTracking):");

var readOnlyProducts = await context.Products

.AsNoTracking()

.Include(p => p.Category)

.ToListAsync();

foreach (var p in readOnlyProducts)

{

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

}

// Compiled query execution

Console.WriteLine("\nCompiled query: Products with price > ₹10000");

var expensiveProducts = await \_expensiveProductsQuery(context, 10000);

foreach (var p in expensiveProducts)

{

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

}

}

}

**OUTPUT**:

Products (read-only using AsNoTracking):

Smartphone - ₹25000 - Category: Electronics

Compiled query: Products with price > ₹10000

Smartphone - ₹25000 - Category: Electronics

**Lab 14: Bulk Operations:**

await context.BulkUpdateAsync(productList);

PROGRAM.CS

using System;

using System.Collections.Generic;

using System.Threading.Tasks;

using Microsoft.EntityFrameworkCore;

using EFCore.BulkExtensions;

class Program

{

static async Task Main()

{

using var context = new AppDbContext();

// Ensure database exists (optional)

await context.Database.EnsureCreatedAsync();

Console.WriteLine("Simulating stock audit - bulk update of stock quantities");

// Load all products

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

// Simulate stock adjustment: +5 units for each product

foreach (var product in allProducts)

{

product.StockQuantity += 5;

}

// Bulk update in one call

await context.BulkUpdateAsync(allProducts);

Console.WriteLine("Bulk stock update completed.");

// Display updated stock

Console.WriteLine("\nUpdated stock quantities:");

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

foreach (var p in updatedProducts)

{

Console.WriteLine($"{p.Name} - Stock: {p.StockQuantity}");

}

}

}

**OUTPUT**;

Simulating stock audit - bulk update of stock quantities

Bulk stock update completed.

Updated stock quantities:

Smartphone - Stock: 55

Wheat Flour - Stock: 105

**Lab 15: Concurrency with RowVersion:**

**Add Property:**

public byte[] RowVersion { get; set; }

**Handle Conflicts:**

try {

await context.SaveChangesAsync();

} catch (DbUpdateConcurrencyException ex) {

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

DbContext:

Console.WriteLine("Simulating concurrency conflict...");

// Load the same product in two separate contexts

var product1 = await context.Products.FirstOrDefaultAsync(p => p.Name == "Smartphone");

using var context2 = new AppDbContext();

var product2 = await context2.Products.FirstOrDefaultAsync(p => p.Name == "Smartphone");

// Update from context1

product1.Price = 26000;

await context.SaveChangesAsync();

Console.WriteLine("First update succeeded (price = 26000).");

// Try to update from context2 — should throw concurrency exception

product2.Price = 24000;

try

{

await context2.SaveChangesAsync();

Console.WriteLine("Second update succeeded (price = 24000).");

}

catch (DbUpdateConcurrencyException)

{

Console.WriteLine("Concurrency conflict detected — update aborted.");

}

**OUTPUT**:

Simulating concurrency conflict...

First update succeeded (price = 26000).

Concurrency conflict detected — update aborted.