**EF Core 8.0 Guided Hands-On Exercises**

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

|  |
| --- |
| //ORM - Object Relational Mapping is a technique that allows developers to interact Relational Databse to Object Orientated Programming  //For example, a C# class Product can be directly mapped to a SQL table Products, it will translate the code to query  //EF Core is a modern, lightweight, cross-platform ORM that supports LINQ, async operations, and compiled queries,  //while EF Framework is limited to Windows and is more suited to legacy .NET applications.  //EF Core 8.0 introduced JSON column mapping, allowing complex data to be stored in a single column |

**A screenshot of a computer

AI-generated content may be incorrect.**

**Lab 2: Setting Up the Database Context for a Retail Store**

**Program.cs**

|  |
| --- |
| var configuration = new ConfigurationBuilder()  .AddJsonFile("appsettings.json", optional: false, reloadOnChange: true)  .Build();  var connectionString = configuration.GetConnectionString("DefaultConnection");  var optionsBuilder = new DbContextOptionsBuilder<AppDbContext>();  optionsBuilder.UseSqlServer(connectionString);  using (var context = new AppDbContext(optionsBuilder.Options))  {  context.Database.EnsureCreated();  Console.WriteLine("Database created successfully!");  } |

**AppDbContext.cs**

|  |
| --- |
| namespace RetailInventory.Data  {  public class AppDbContext : DbContext  {  public AppDbContext(DbContextOptions<AppDbContext> options)  : base(options)  {  }  public DbSet<Product> Products { get; set; }  public DbSet<Category> Categories { get; set; }  }  } |

**Appsettings.json**

|  |
| --- |
| {  "ConnectionStrings": {  "DefaultConnection": "Server=NEERAJ\\SQLEXPRESS;Database=RetailDb;Trusted\_Connection=True;Encrypt=False;"  }  } |

**A screenshot of a computer

AI-generated content may be incorrect.**

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

**20250706070702\_InitialCreate.cs**

|  |
| --- |
| **using Microsoft.EntityFrameworkCore.Migrations;**  **#nullable disable**  **namespace RetailInventory.Migrations**  **{**  **/// <inheritdoc />**  **public partial class InitialCreate : Migration**  **{**  **/// <inheritdoc />**  **protected override void Up(MigrationBuilder migrationBuilder)**  **{**  **migrationBuilder.CreateTable(**  **name: "Categories",**  **columns: table => new**  **{**  **Id = table.Column<int>(type: "int", nullable: false)**  **.Annotation("SqlServer:Identity", "1, 1"),**  **Name = table.Column<string>(type: "nvarchar(max)", nullable: false)**  **},**  **constraints: table =>**  **{**  **table.PrimaryKey("PK\_Categories", x => x.Id);**  **});**  **migrationBuilder.CreateTable(**  **name: "Products",**  **columns: table => new**  **{**  **Id = table.Column<int>(type: "int", nullable: false)**  **.Annotation("SqlServer:Identity", "1, 1"),**  **Name = table.Column<string>(type: "nvarchar(max)", nullable: false),**  **Price = table.Column<decimal>(type: "decimal(18,2)", nullable: false),**  **CategoryId = table.Column<int>(type: "int", nullable: false)**  **},**  **constraints: table =>**  **{**  **table.PrimaryKey("PK\_Products", x => x.Id);**  **table.ForeignKey(**  **name: "FK\_Products\_Categories\_CategoryId",**  **column: x => x.CategoryId,**  **principalTable: "Categories",**  **principalColumn: "Id",**  **onDelete: ReferentialAction.Cascade);**  **});**  **migrationBuilder.CreateIndex(**  **name: "IX\_Products\_CategoryId",**  **table: "Products",**  **column: "CategoryId");**  **}**  **/// <inheritdoc />**  **protected override void Down(MigrationBuilder migrationBuilder)**  **{**  **migrationBuilder.DropTable(**  **name: "Products");**  **migrationBuilder.DropTable(**  **name: "Categories");**  **}**  **}**  **}** |

**A screenshot of a computer

AI-generated content may be incorrect.**

**Lab 4: Inserting Initial Data into the Database**

**Program.cs**

|  |
| --- |
| using Microsoft.EntityFrameworkCore;  using Microsoft.Extensions.Configuration;  using RetailInventory.Data;  using RetailInventory.Models;  using System.Threading.Tasks;  var configuration = new ConfigurationBuilder()  .AddJsonFile("appsettings.json", optional: false, reloadOnChange: true)  .Build();  var connectionString = configuration.GetConnectionString("DefaultConnection");  var optionsBuilder = new DbContextOptionsBuilder<AppDbContext>();  optionsBuilder.UseSqlServer(connectionString);  using var context = new AppDbContext(optionsBuilder.Options);  await SeedDataAsync(context);  static async Task SeedDataAsync(AppDbContext context)  {  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();  } |

**A screenshot of a computer screen

AI-generated content may be incorrect.**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Lab 5: Retrieving Data from the Database**

**Program.cs**

|  |
| --- |
| **using Microsoft.EntityFrameworkCore;**  **using Microsoft.Extensions.Configuration;**  **using RetailInventory.Data;**  **using RetailInventory.Models;**  **using System.Threading.Tasks;**  **var configuration = new ConfigurationBuilder()**  **.AddJsonFile("appsettings.json", optional: false, reloadOnChange: true)**  **.Build();**  **var connectionString = configuration.GetConnectionString("DefaultConnection");**  **var optionsBuilder = new DbContextOptionsBuilder<AppDbContext>();**  **optionsBuilder.UseSqlServer(connectionString);**  **using var context = new AppDbContext(optionsBuilder.Options);**  **await RetriveDataAsync(context);**  **static async Task SeedDataAsync(AppDbContext context)**  **{**  **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();**  **}**  **static async Task RetriveDataAsync(AppDbContext context)**  **{**  **var products = await context.Products.ToListAsync();**  **foreach (var p in products)**  **Console.WriteLine($"{p.Name} - {p.Price}");**  **var product = await context.Products.FindAsync(1);**  **Console.WriteLine($"Found: {product?.Name}");**  **var expensive = await context.Products.FirstOrDefaultAsync(p => p.Price > 50000);**  **Console.WriteLine($"Expensive: {expensive?.Name}");**  **}** |

**A screenshot of a computer

AI-generated content may be incorrect.**

**Lab 6: Updating and Deleting Records**

**Program.cs**

|  |
| --- |
| using Microsoft.EntityFrameworkCore;  using Microsoft.Extensions.Configuration;  using RetailInventory.Data;  using RetailInventory.Models;  using System.Threading.Tasks;  var configuration = new ConfigurationBuilder()  .AddJsonFile("appsettings.json", optional: false, reloadOnChange: true)  .Build();  var connectionString = configuration.GetConnectionString("DefaultConnection");  var optionsBuilder = new DbContextOptionsBuilder<AppDbContext>();  optionsBuilder.UseSqlServer(connectionString);  using var context = new AppDbContext(optionsBuilder.Options);  await UpdateDataAsync(context);  static async Task SeedDataAsync(AppDbContext context)  {  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();  }  static async Task RetriveDataAsync(AppDbContext context)  {  var products = await context.Products.ToListAsync();  foreach (var p in products)  Console.WriteLine($"{p.Name} - {p.Price}");  var product = await context.Products.FindAsync(1);  Console.WriteLine($"Found: {product?.Name}");  var expensive = await context.Products.FirstOrDefaultAsync(p => p.Price > 50000);  Console.WriteLine($"Expensive: {expensive?.Name}");  }  static async Task UpdateDataAsync(AppDbContext context)  {  var product = await context.Products.FirstOrDefaultAsync(p => p.Name == "Laptop");  if (product != null)  {  product.Price = 70000;  await context.SaveChangesAsync();  }  }  static async Task DeleteDataAsync(AppDbContext context)  {  var toDelete = await context.Products.FirstOrDefaultAsync(p => p.Name == "Rice Bag");  if (toDelete != null)  {  context.Products.Remove(toDelete);  await context.SaveChangesAsync();  }  } |

**A screenshot of a computer screen

AI-generated content may be incorrect.**

**Program.cs**

|  |
| --- |
| using Microsoft.EntityFrameworkCore;  using Microsoft.Extensions.Configuration;  using RetailInventory.Data;  using RetailInventory.Models;  using System.Threading.Tasks;  var configuration = new ConfigurationBuilder()  .AddJsonFile("appsettings.json", optional: false, reloadOnChange: true)  .Build();  var connectionString = configuration.GetConnectionString("DefaultConnection");  var optionsBuilder = new DbContextOptionsBuilder<AppDbContext>();  optionsBuilder.UseSqlServer(connectionString);  using var context = new AppDbContext(optionsBuilder.Options);  await DeleteDataAsync(context);  static async Task SeedDataAsync(AppDbContext context)  {  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();  }  static async Task RetriveDataAsync(AppDbContext context)  {  var products = await context.Products.ToListAsync();  foreach (var p in products)  Console.WriteLine($"{p.Name} - {p.Price}");  var product = await context.Products.FindAsync(1);  Console.WriteLine($"Found: {product?.Name}");  var expensive = await context.Products.FirstOrDefaultAsync(p => p.Price > 50000);  Console.WriteLine($"Expensive: {expensive?.Name}");  }  static async Task UpdateDataAsync(AppDbContext context)  {  var product = await context.Products.FirstOrDefaultAsync(p => p.Name == "Laptop");  if (product != null)  {  product.Price = 70000;  await context.SaveChangesAsync();  }  }  static async Task DeleteDataAsync(AppDbContext context)  {  var toDelete = await context.Products.FirstOrDefaultAsync(p => p.Name == "Rice Bag");  if (toDelete != null)  {  context.Products.Remove(toDelete);  await context.SaveChangesAsync();  }  } |

**A screenshot of a computer screen

AI-generated content may be incorrect.**

**Lab 7: Writing Queries with LINQ**

**Program.cs**

|  |
| --- |
| using Microsoft.EntityFrameworkCore;  using Microsoft.Extensions.Configuration;  using RetailInventory.Data;  using RetailInventory.Models;  using System.Threading.Tasks;  var configuration = new ConfigurationBuilder()  .AddJsonFile("appsettings.json", optional: false, reloadOnChange: true)  .Build();  var connectionString = configuration.GetConnectionString("DefaultConnection");  var optionsBuilder = new DbContextOptionsBuilder<AppDbContext>();  optionsBuilder.UseSqlServer(connectionString);  using var context = new AppDbContext(optionsBuilder.Options);  await LinqQueriesAsync(context);  static async Task SeedDataAsync(AppDbContext context)  {  var electronics = await context.Categories.FirstOrDefaultAsync(c => c.Name == "Electronics");  var groceries = await context.Categories.FirstOrDefaultAsync(c => c.Name == "Groceries");  var product1 = new Product  {  Name = "Bluetooth Headphone",  Price = 5000,  Category = electronics  };  var product2 = new Product  {  Name = "Ghee",  Price = 600,  Category = groceries  };  await context.Products.AddRangeAsync(product1, product2);  await context.SaveChangesAsync();  Console.WriteLine("Products inserted successfully.");  }  static async Task RetriveDataAsync(AppDbContext context)  {  var products = await context.Products.ToListAsync();  foreach (var p in products)  Console.WriteLine($"{p.Name} - {p.Price}");  var product = await context.Products.FindAsync(1);  Console.WriteLine($"Found: {product?.Name}");  var expensive = await context.Products.FirstOrDefaultAsync(p => p.Price > 50000);  Console.WriteLine($"Expensive: {expensive?.Name}");  }  static async Task UpdateDataAsync(AppDbContext context)  {  var product = await context.Products.FirstOrDefaultAsync(p => p.Name == "Laptop");  if (product != null)  {  product.Price = 70000;  await context.SaveChangesAsync();  }  }  static async Task DeleteDataAsync(AppDbContext context)  {  var toDelete = await context.Products.FirstOrDefaultAsync(p => p.Name == "Electronics");  if (toDelete != null)  {  context.Products.Remove(toDelete);  await context.SaveChangesAsync();  }  }  static async Task LinqQueriesAsync(AppDbContext context)  {  var filtered = await context.Products  .Where(p => p.Price > 1000)  .OrderByDescending(p => p.Price)  .ToListAsync();  Console.WriteLine("Filtered Product Based on thier Price");  foreach(var p in filtered)  Console.WriteLine($"{p.Name} - {p.Price}");  Console.WriteLine("\n");  var productDTOs = await context.Products  .Select(p => new { p.Name, p.Price })  .ToListAsync();  Console.WriteLine("DTO Examples");  foreach (var dto in productDTOs)  {  Console.WriteLine($"{dto.Name} - {dto.Price}");  }  } |

**A screenshot of a computer

AI-generated content may be incorrect.**

**Lab 8: Managing Migrations and Schema Changes**

|  |
| --- |
| using Microsoft.EntityFrameworkCore;  using Microsoft.Extensions.Configuration;  using RetailInventory.Data;  using RetailInventory.Models;  using System.Threading.Tasks;  var configuration = new ConfigurationBuilder()  .AddJsonFile("appsettings.json", optional: false, reloadOnChange: true)  .Build();  var connectionString = configuration.GetConnectionString("DefaultConnection");  var optionsBuilder = new DbContextOptionsBuilder<AppDbContext>();  optionsBuilder.UseSqlServer(connectionString);  using var context = new AppDbContext(optionsBuilder.Options);  await UpdateDataAsync(context);  static async Task SeedDataAsync(AppDbContext context)  {  var electronics = await context.Categories.FirstOrDefaultAsync(c => c.Name == "Electronics");  var groceries = await context.Categories.FirstOrDefaultAsync(c => c.Name == "Groceries");  var product1 = new Product  {  Name = "Mechanical Keyboard",  Price = 5000,  StockQuantity = 37,  Category = electronics  };  var product2 = new Product  {  Name = "Butter",  Price = 600,  StockQuantity=200,  Category = groceries  };  await context.Products.AddRangeAsync(product1, product2);  await context.SaveChangesAsync();  Console.WriteLine("Products inserted successfully.");  }  static async Task RetriveDataAsync(AppDbContext context)  {  var products = await context.Products.ToListAsync();  foreach (var p in products)  Console.WriteLine($"{p.Name} - {p.Price}");  var product = await context.Products.FindAsync(1);  Console.WriteLine($"Found: {product?.Name}");  var expensive = await context.Products.FirstOrDefaultAsync(p => p.Price > 50000);  Console.WriteLine($"Expensive: {expensive?.Name}");  }  static async Task UpdateDataAsync(AppDbContext context)  {  var product = await context.Products.FirstOrDefaultAsync(p => p.Name == "Ghee");  if (product != null)  {  product.StockQuantity = 800;  await context.SaveChangesAsync();  }  }  static async Task DeleteDataAsync(AppDbContext context)  {  var toDelete = await context.Products.FirstOrDefaultAsync(p => p.Name == "Electronics");  if (toDelete != null)  {  context.Products.Remove(toDelete);  await context.SaveChangesAsync();  }  }  static async Task LinqQueriesAsync(AppDbContext context)  {  var filtered = await context.Products  .Where(p => p.Price > 1000)  .OrderByDescending(p => p.Price)  .ToListAsync();  Console.WriteLine("Filtered Product Based on thier Price");  foreach(var p in filtered)  Console.WriteLine($"{p.Name} - {p.Price}");  Console.WriteLine("\n");  var productDTOs = await context.Products  .Select(p => new { p.Name, p.Price })  .ToListAsync();  Console.WriteLine("DTO Examples");  foreach (var dto in productDTOs)  {  Console.WriteLine($"{dto.Name} - {dto.Price}");  }  } |

**A table with numbers and text

AI-generated content may be incorrect.**