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

// ORM maps C# classes to relational tables allowing automatic SQL generation. EF Core improves productivity with features like async LINQ, compiled queries, and JSON mapping.

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

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; }

}

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: EF Core CLI to Create and Apply Migrations**

// Commands:

// dotnet tool install --global dotnet-ef

// dotnet ef migrations add InitialCreate

// dotnet ef database update

**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();

**Lab 5: Retrieving Data**

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

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

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

**Lab 6: Updating and Deleting**

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

if (p != null) { p.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(); }

**Lab 7: LINQ Queries**

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();

**Lab 8: Schema Changes (Add StockQuantity)**

public class Product

{

public int Id { get; set; }

public string Name { get; set; }

public decimal Price { get; set; }

public int StockQuantity { get; set; } // new

}

// Commands:

// dotnet ef migrations add AddStockQuantity

// dotnet ef database update

**Lab 9: Seeding Data**

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

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 });

}

**Lab 10: Eager, Explicit, Lazy Loading**

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

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

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

**Lab 11: Relationships**

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);

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; } }

**Lab 12: DTO for Circular Reference**

public class ProductDTO { public string Name { get; set; } public string CategoryName { get; set; } }

var dtos = await context.Products.Select(p => new ProductDTO { Name = p.Name, CategoryName = p.Category.Name }).ToListAsync();

**Lab 13: Query Caching and NoTracking**

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

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

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

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

var result = await expensiveQuery(context, 10000);

**Lab 14: Bulk Update**

await context.BulkUpdateAsync(productList);

**Lab 15: Concurrency**

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

try { await context.SaveChangesAsync(); }

catch (DbUpdateConcurrencyException) { Console.WriteLine("Conflict Detected"); }