EF Core 8.0 Labs 1 to 5 - Assignment Solution

# Lab 1: Understanding ORM

\*\*What is ORM?\*\*  
ORM (Object-Relational Mapping) maps C# classes to SQL tables.  
Example:  
```csharp  
public class Product {  
 public int Id { get; set; }  
 public string Name { get; set; }  
}  
```  
Maps to SQL:  
```sql  
CREATE TABLE Products (  
 Id INT PRIMARY KEY,  
 Name NVARCHAR(MAX)  
);  
```

\*\*Benefits:\*\*  
- Avoids raw SQL  
- Faster development  
- Maintains sync between code and DB

\*\*EF Core vs EF Framework:\*\*  
- EF Core: Cross-platform, modern, lightweight  
- EF Framework: Windows-only, mature, but less flexible

# Lab 2: Setting Up Database Context

\*\*Step 1: Create a Console App:\*\*  
```bash  
dotnet new console -n RetailInventory  
cd RetailInventory  
```

\*\*Step 2: Install EF Core Packages:\*\*  
```bash  
dotnet add package Microsoft.EntityFrameworkCore.SqlServer  
dotnet add package Microsoft.EntityFrameworkCore.Design  
```

\*\*Step 3: Create Models:\*\*  
Category.cs  
```csharp  
public class Category {  
 public int Id { get; set; }  
 public string Name { get; set; }  
 public List<Product> Products { get; set; }  
}  
```  
Product.cs  
```csharp  
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; }  
}  
```

\*\*Step 4: Create AppDbContext.cs:\*\*  
```csharp  
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 CLI & Migrations

\*\*Step 1: Install EF CLI:\*\*  
```bash  
dotnet tool install --global dotnet-ef  
```

\*\*Step 2: Create Migration:\*\*  
```bash  
dotnet ef migrations add InitialCreate  
```

\*\*Step 3: Update Database:\*\*  
```bash  
dotnet ef database update  
```

# Lab 4: Inserting Initial Data

\*\*Program.cs Code to Insert Data:\*\*  
```csharp  
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();  
```

\*\*Run the app:\*\*  
```bash  
dotnet run  
```

# Lab 5: Retrieving Data

\*\*Retrieve Data:\*\*  
```csharp  
var context = new AppDbContext();  
  
// All products  
var products = await context.Products.ToListAsync();  
foreach (var p in products)  
 Console.WriteLine($"{p.Name} - ₹{p.Price}");  
  
// Find by ID  
var product = await context.Products.FindAsync(1);  
Console.WriteLine($"Found: {product?.Name}");  
  
// Expensive product  
var expensive = await context.Products.FirstOrDefaultAsync(p => p.Price > 50000);  
Console.WriteLine($"Expensive: {expensive?.Name}");  
```