Lab 4: Inserting Initial Data into the Database

# Introduction

After designing your database schema and creating it using EF Core migrations, the next step is to populate your database with initial data. This is often called data seeding and is essential for testing, demonstration, or providing default values for your application.  
  
In this lab, you will use EF Core’s asynchronous methods (AddRangeAsync, SaveChangesAsync) to insert initial records into your database.

# Objectives

- Understand how to insert data using EF Core in a .NET application.  
- Learn to use asynchronous methods for database operations.  
- Verify inserted data using SQL Server tools.

# Scenario

The store manager wants to add initial product categories and products to the system. You will use EF Core to insert this data into the SQL Server database.

# Step 1: Insert Data in Program.cs

Below is a sample Program.cs for inserting initial data. Note: Adjust property names if your Category or Product models differ.

using System;  
using System.Threading.Tasks;  
  
class Program  
{  
 static async Task Main(string[] args)  
 {  
 using var context = new AppDbContext();  
  
 // Create categories  
 var electronics = new Category { Name = "Electronics" };  
 var groceries = new Category { Name = "Groceries" };  
  
 // Add categories to the database  
 await context.Categories.AddRangeAsync(electronics, groceries);  
  
 // Create products and assign categories  
 var product1 = new Product { Name = "Laptop", Price = 75000, Category = electronics };  
 var product2 = new Product { Name = "Rice Bag", Price = 1200, Category = groceries };  
  
 // Add products to the database  
 await context.Products.AddRangeAsync(product1, product2);  
  
 // Save all changes to the database  
 await context.SaveChangesAsync();  
  
 Console.WriteLine("Initial data inserted successfully!");  
 }  
}

## Key Points:

- The using var context = new AppDbContext(); ensures proper disposal of the database context.  
- AddRangeAsync is used to add multiple entities efficiently.  
- SaveChangesAsync commits all changes to the database.  
- The async Task Main method enables asynchronous programming in the entry point.

# Step 2: Run the Application

Open your terminal in the project directory and run:

dotnet run

You should see:

Initial data inserted successfully!

# Step 3: Verify the Data in SQL Server

Open SQL Server Management Studio (SSMS) or Azure Data Studio.  
  
Connect to your database (e.g., RetailInventoryDb).  
  
Run the following queries:

SELECT \* FROM Categories;  
SELECT \* FROM Products;

Check the results:  
  
- The Categories table should contain "Electronics" and "Groceries".  
- The Products table should contain "Laptop" (linked to Electronics) and "Rice Bag" (linked to Groceries).

# Best Practices

- Always use SaveChangesAsync for non-blocking database operations in modern .NET applications.  
- Use navigation properties (e.g., Category in Product) to simplify relationships.  
- Insert parent entities (categories) before child entities (products) to ensure foreign key relationships are valid.  
- Use AddRangeAsync for efficient bulk inserts.

# Troubleshooting Tips

- If you get errors about missing properties, ensure your model classes match the code.  
- If you see duplicate data after running the app multiple times, clear the tables or add logic to check for existing data before inserting.  
- If you get database connection errors, verify your connection string and that your SQL Server instance is running.