1. What is ORM?

- **ORM** stands for **Object-Relational Mapping**.
- It connects your **C# classes (objects)** to **SQL tables** in a database.

📌 **Example**:
If you have a class like:

```
public class Product {
    public int Id { get; set; }
    public string Name { get; set; }
}
```

EF Core will automatically create a `Products` table in SQL Server with `Id` and `Name` columns.

🌟 Benefits of ORM:

- **Productivity**: Less manual SQL writing.
- **Maintainability**: Code and database stay in sync.
- **Abstraction**: You work with C# code, not SQL queries.

✅ 2. EF Core vs EF Framework

Feature	EF Core	EF Framework (EF6)
Platform	Cross-platform (.NET Core)	Windows only (.NET Framework)
Modern Features	✅ LINQ, async, compiled queries	❌ Limited async support
Lightweight	✅ Yes	❌ No
Maturity	Still evolving	More stable/older version

✅ 3. EF Core 8.0 New Features

- **JSON column mapping**: Store and read JSON directly in the database.
- **Compiled models**: Speeds up performance.
- **Interceptors**: Customize database behavior.
- **Better bulk operations**: Insert/update many records faster.

✅ 4. Create .NET Console App

```
dotnet new console -n RetailInventory
cd RetailInventory
```

✅ 5. Install EF Core Packages

```
dotnet add package Microsoft.EntityFrameworkCore.SqlServer
dotnet add package Microsoft.EntityFrameworkCore.Design
```

🧪 Lab 2: Setting Up the Database Context

✅ 1. Create Models

```
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; }
}
```

✅ 2. Create AppDbContext

```
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.UseSqlServer("Your_Connection_String_Here");
    }
}
```

💡 Replace `"Your_Connection_String_Here"` with your actual SQL Server connection string.

🔧 Lab 3: Creating and Applying Migrations

✅ 1. Install EF CLI

```
dotnet tool install --global dotnet-ef
```

✅ 2. Create Migration

```
dotnet ef migrations add InitialCreate
```

✅ 3. Apply Migration

```
dotnet ef database update
```

✅ 4. Verify in SQL Server

Check that the `Products` and `Categories` tables are created.

📥 Lab 4: Inserting Initial Data

✅ 1. Insert Data in `Program.cs`

```
using System;
using System.Threading.Tasks;

class Program {
    static async Task Main(string[] args) {
        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();

        Console.WriteLine("Data inserted!");
    }
}
```

✅ 2. Run the App

```
dotnet run
```

✅ 3. Check SQL Server

Open SQL Server Management Studio (SSMS) to see if the data is there.

🔍 Lab 5: Retrieving Data from the Database

✅ 1. Get All Products

```
var products = await context.Products.ToListAsync();
foreach (var p in products)
    Console.WriteLine($"{p.Name} - ₹{p.Price}");
```

✅ 2. Find Product by ID

```
var product = await context.Products.FindAsync(1);
Console.WriteLine($"Found: {product?.Name}");
```

✅ 3. Get First Expensive Product

```
var expensive = await context.Products.FirstOrDefaultAsync(p =>
p.Price > 50000);
Console.WriteLine($"Expensive: {expensive?.Name}");
```
