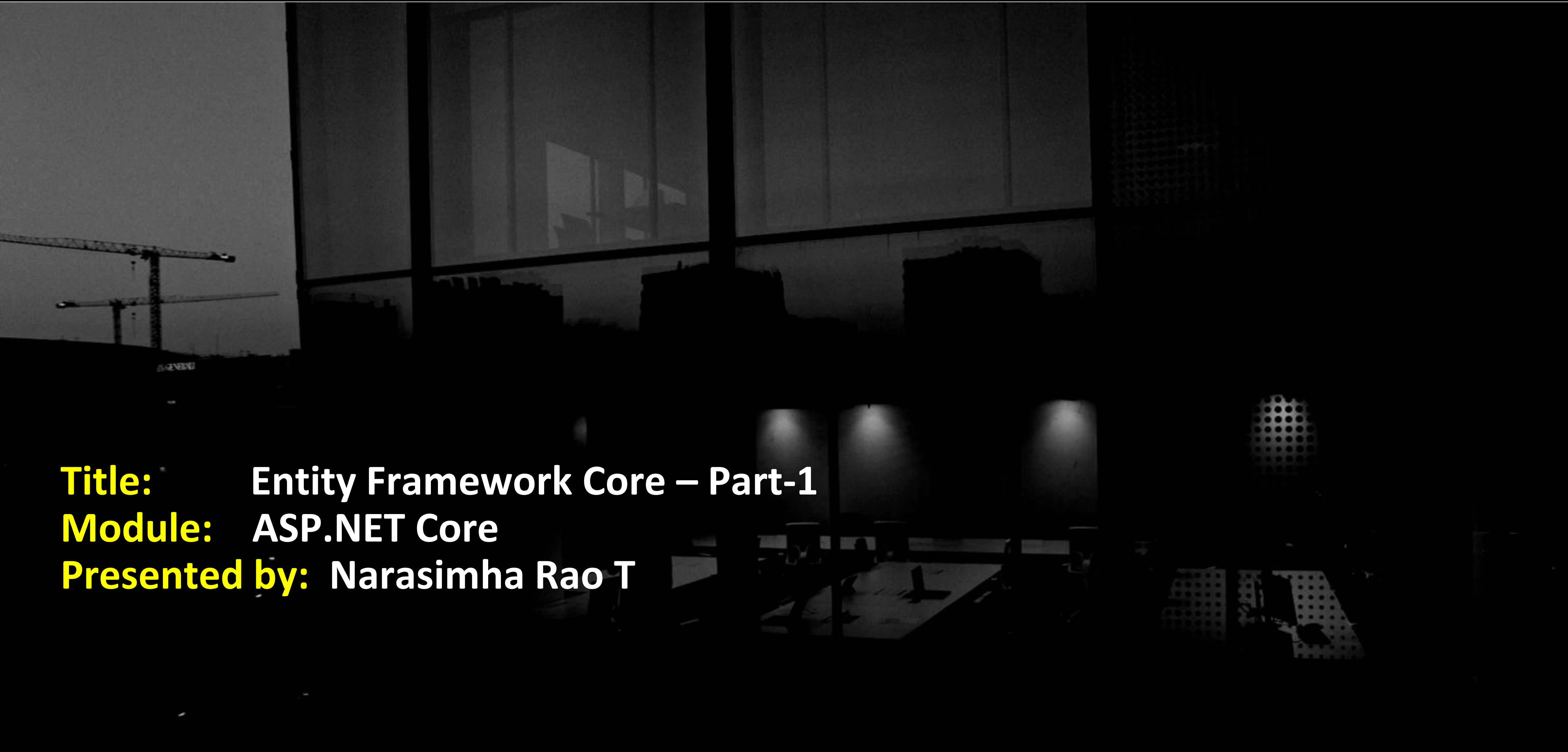


Title: Entity Framework Core – Part-1

Module: ASP.NET Core

Presented by: Narasimha Rao T

Entity Framework Core in ASP.NET Core - Part-1

By

Narasimha Rao T

Microsoft.Net FSD Trainer

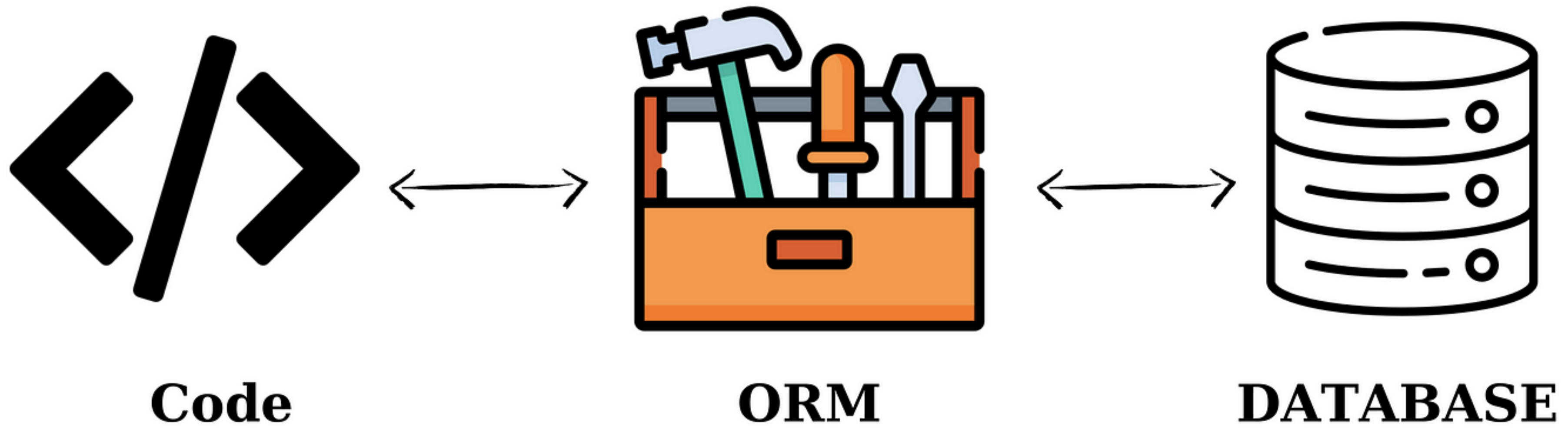
Professional Development Trainer

tnrao.trainer@gmail.com

What is ORM?

1. What is ORM?

- **ORM (Object Relational Mapping):**
  - A technique to map **objects in code (classes)** to **database tables**.
  - Simplifies interaction with relational databases by avoiding raw SQL queries.
  - Allows developers to work with **objects and LINQ queries** instead of SQL.


Code
ORM
DATABASE

Advantages of ORM:

- Productivity: Less SQL writing.
- Maintainability: Clean, object-oriented code.
- Database independence: Switch between DB providers with minimal changes.
- Security: Reduces risk of SQL injection when using parameterized queries.

2. Examples of ORM Tools

- **Entity Framework Core (EF Core)** → Microsoft’s ORM for .NET.
- **NHibernate** → Mature ORM for .NET.
- **Dapper** → Lightweight micro-ORM (focuses on performance).
- **LLBLGen Pro, Telerik OpenAccess** → Other commercial ORMs.

Entity Framework Core

3. Introduction to Entity Framework Core

- EF Core = **Modern, lightweight, cross-platform ORM**.
- Supports:
    - LINQ queries
    - Change tracking
    - Migrations
    - Database providers (SQL Server, SQLite, PostgreSQL, MySQL, etc.)
- Works with **ASP.NET Core, Console Apps, Blazor, WPF** etc.

4. Overview and Installation of Packages for EF Core

- Install **NuGet packages** in ASP.NET Core project:

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

- **Common Packages:**
    - `Microsoft.EntityFrameworkCore` → Base package.
    - `Microsoft.EntityFrameworkCore.SqlServer` → SQL Server provider.
    - `Microsoft.EntityFrameworkCore.Sqlite` → SQLite.
    - `Microsoft.EntityFrameworkCore.Tools` → Migration/scaffolding commands.

5. DbContext and Entity Classes

- **Entity Class** = Represents a table.

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

- **DbContext** = Bridge between C# classes & DB.

```csharp
public class AppDbContext : DbContext
{
    public DbSet<Student> Students { get; set; }
    public AppDbContext(DbContextOptions<AppDbContext> options) : base(options) { }
}
```

6. Code-First Approach

- Start with **C# classes**, then generate DB schema.
- Steps:
    i. Define entity classes.
    ii. Define `DbContext`.
    iii. Configure connection string in `appsettings.json`.
    iv. Run migrations to create/update DB.

7. Migrations

- EF Core generates & applies schema changes.

Commands:

- `Add-Migration MigrationName` → Create migration script.
- `Update-Database` → Apply migrations to DB.
- `Remove-Migration` → Undo last migration.

Perform CRUD Operations using EF Core

Async Methods to perform CRUD

- Prefer `async/await` in ASP.NET Core apps:
    - `AddAsync()`
    - `FindAsync()`
    - `FirstOrDefaultAsync()`
    - `SaveChangesAsync()`

8. DB-First Approach using Scaffolding

- Start with **existing database**, then generate models & context.
- Use command:

```
Scaffold-DbContext "connection_string" Microsoft.EntityFrameworkCore.SqlServer -OutputDir Models
```

9. What is DB-First vs. Code-First?

- **Code-First:** Start with classes → Generate DB.
- **DB-First:** Start with DB → Generate classes.
- Choice depends on:
    - New project → Code-First.
    - Existing DB → DB-First.

10. Scaffold from Existing DB using Scaffold-DbContext

- Example command:

```
Scaffold-DbContext "Server=.;Database=SchoolDb;Trusted_Connection=True;" Microsoft.EntityFrameworkCore.SqlServer -OutputDir Models
```

- Options:
  - `-Context` → Custom DbContext name.
  - `-OutputDir` → Output directory for entities.
  - `-Schemas` → Include only specific schemas.
  - `-Tables` → Scaffold specific tables.
  - `-DataAnnotations` → Use attributes instead of Fluent API.

11. Connection String, Provider, Output Directory, Pluralization Settings

- **Connection String:** In `appsettings.json`

```
"ConnectionStrings": {
  "DefaultConnection": "Server=.;Database=SchoolDb;Trusted_Connection=True;"
}
```

- **Provider:** (SQL Server, MySQL, etc.)
- **Output Directory:** Use `-OutputDir Models`.
- **Pluralization:** By default EF Core pluralizes table names → Can disable in `OnModelCreating`.

12. Entity-Specific Scaffolding

- Scaffold only selected tables:

```
Scaffold-DbContext "connection_string" Microsoft.EntityFrameworkCore.SqlServer -OutputDir Models -Tables Student,Course
```

13. Clean-up Tips after Scaffolding

- Remove unwanted navigation properties.
- Rename generated classes for better readability.
- Move `DbContext` to separate folder.
- Add partial classes for customization (avoid editing auto-generated code directly).

Quiz Time

14. Some Interview Questions

1. What is ORM, and why do we use it?
2. Difference between EF Core and ADO.NET?
3. Explain Code-First vs DB-First approaches.
4. What are Migrations in EF Core?
5. How does EF Core handle relationships (1-1, 1-many, many-many)?
6. What is Lazy Loading vs Eager Loading in EF Core?
7. How do you optimize EF Core performance?
8. What happens when you call `SaveChanges()` in EF Core?
9. What are shadow properties in EF Core?
10. How do you scaffold only specific tables from DB?