# Entity Framework Core

## Agenda

- What is EF Core?
- Key concepts
- Querying data
- Saving data
- Apply EF Core to project
- Seeding data

## What is EF Core?

- Entity Framework (EF) Core is a lightweight, extensible, open source and cross-platform data access technology.
- EF Core can serve as an object-relational mapper (O/RM), which:
  - Enables .NET developers to work with a database using .NET objects.
  - Eliminates the need for most of the data-access code that typically needs to be written.
  - Supports many database engines.

Entity Framework Core

3

## The model

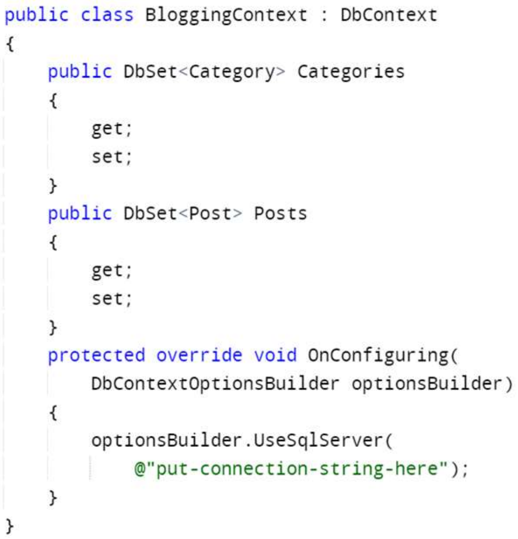
- With EF Core, data access is performed using a model.
- A model is made up of:
  - Entity classes: Represents the structure of database tables or views. These classes must be included as a DbSet<TEntity> type property in the DbContext class.
  - A context object: Represents a session with the database. The context object allows querying and saving data.

4

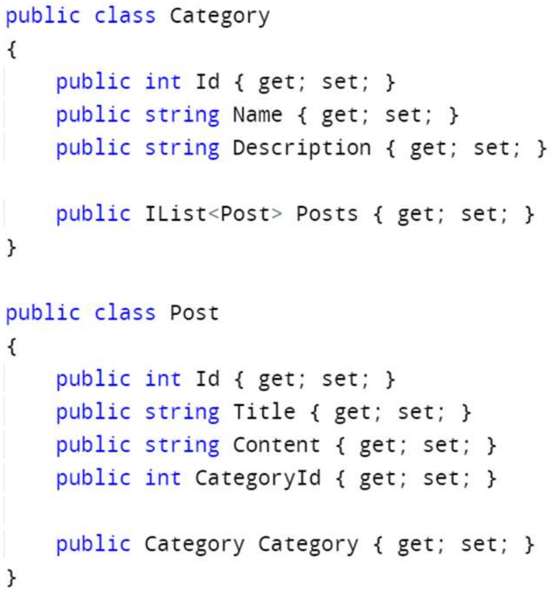
# Model development approaches

- Database first
  - A database already exists.
  - Generate a model from an existing database using the Reverse Engineer feature.
- Code first:
  - No database exists.
  - Hand code to build a model.
  - Use EF Migrations to create a database from the model that matches its structure and features.

# Example



```
public class BloggingContext : DbContext
{
    public DbSet<Category> Categories
    {
        get;
        set;
    }
    public DbSet<Post> Posts
    {
        get;
        set;
    }
    protected override void OnConfiguring(
        DbContextOptionsBuilder optionsBuilder)
    {
        optionsBuilder.UseSqlServer(
            @"put-connection-string-here");
    }
}
```

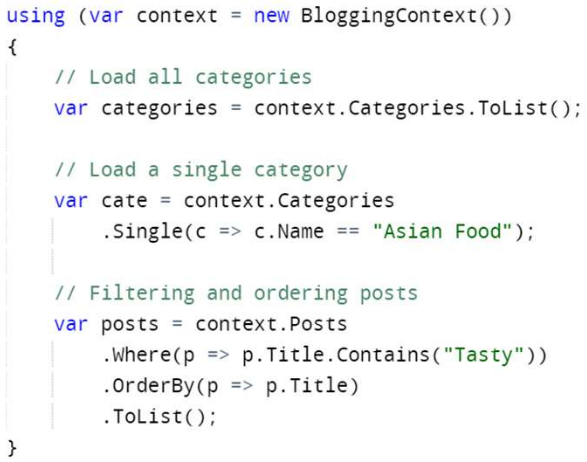


```
public class Category
{
    public int Id { get; set; }
    public string Name { get; set; }
    public string Description { get; set; }

    public IList<Post> Posts { get; set; }
}

public class Post
{
    public int Id { get; set; }
    public string Title { get; set; }
    public string Content { get; set; }
    public int CategoryId { get; set; }

    public Category Category { get; set; }
}
```

ITDLU

# Querying Data

- EF Core uses Language-Integrated Query (LINQ) to query data from the database.
- LINQ allows you to use C# to write strongly typed queries.
- It uses your derived context and entity classes to reference database objects.
- EF Core passes a representation of the LINQ query to the database provider.
- Database providers in turn translate it to database-specific query language.



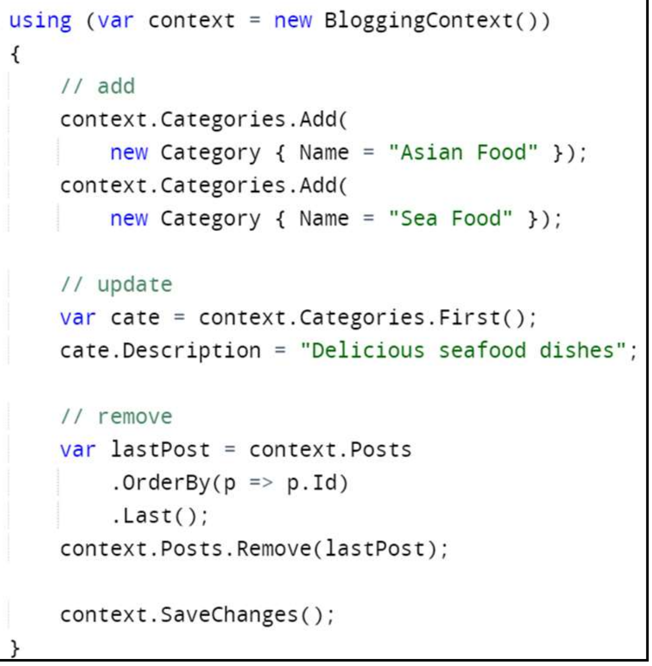
```
using (var context = new BloggingContext())
{
    // Load all categories
    var categories = context.Categories.ToList();

    // Load a single category
    var cate = context.Categories
        .Single(c => c.Name == "Asian Food");

    // Filtering and ordering posts
    var posts = context.Posts
        .Where(p => p.Title.Contains("Tasty"))
        .OrderBy(p => p.Title)
        .ToList();
}
```

7

ITDLU

# Saving Data

- Data is created, deleted, and modified in the database using instances of your entity classes.
- Each context instance has a ChangeTracker that is responsible for keeping track of changes that need to be written to the database.
- The database provider is responsible for translating the changes into database-specific operations (INSERT, UPDATE, and DELETE commands for a relational database).



```
using (var context = new BloggingContext())
{
    // add
    context.Categories.Add(
        new Category { Name = "Asian Food" });
    context.Categories.Add(
        new Category { Name = "Sea Food" });

    // update
    var cate = context.Categories.First();
    cate.Description = "Delicious seafood dishes";

    // remove
    var lastPost = context.Posts
        .OrderBy(p => p.Id)
        .Last();
    context.Posts.Remove(lastPost);

    context.SaveChanges();
}
```

8

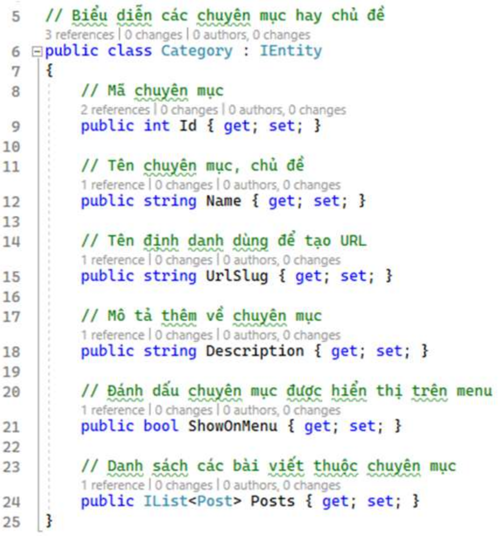
# Apply EF Core Code First

- Install EF Core
- Create model
  - Define entity types
  - Configure entities
  - Define context class
  - Configure connection string
- Add initial data (data seeding)
- Create database using EF Migrations
- Build business logic layer

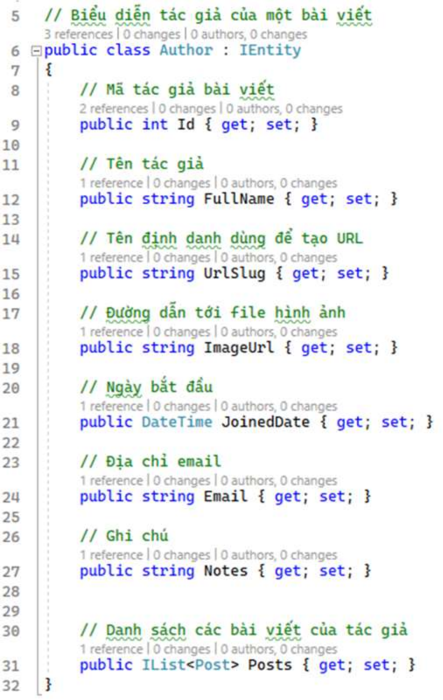
# Install EF Core

- Use one of following methods:
  - .NET Core command-line interface (CLI)
  - Visual Studio Package Manager Dialog
  - Visual Studio Package Manager Console
- NuGet packages:
  - Microsoft.EntityFrameworkCore
  - Microsoft.EntityFrameworkCore.SqlServer (SQL Server provider)
  - Microsoft.EntityFrameworkCore.Tools (Tools for NPM console in VS)
- To run migration from .NET CLI, .NET EF tool must be installed
  - `dotnet tool install --global dotnet-ef`

# Define entity types



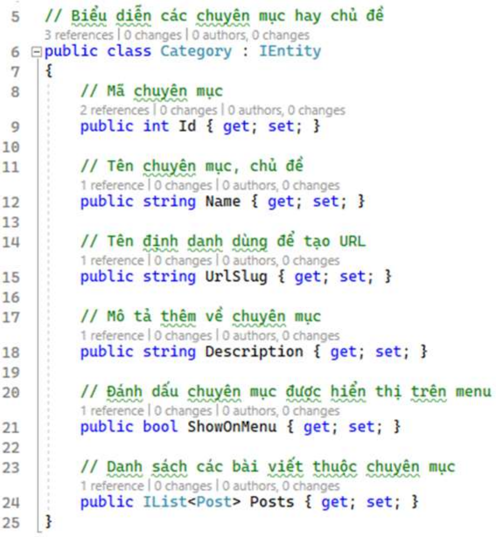
```
// Biểu diễn các chuyên mục hay chủ đề
3 references | 0 changes | 0 authors, 0 changes
public class Category : IEntity
{
    // Mã chuyên mục
    2 references | 0 changes | 0 authors, 0 changes
    public int Id { get; set; }

    // Tên chuyên mục, chủ đề
    1 reference | 0 changes | 0 authors, 0 changes
    public string Name { get; set; }

    // Tên định danh dùng để tạo URL
    1 reference | 0 changes | 0 authors, 0 changes
    public string UrlSlug { get; set; }

    // Mô tả thêm về chuyên mục
    1 reference | 0 changes | 0 authors, 0 changes
    public string Description { get; set; }

    // Đánh dấu chuyên mục được hiển thị trên menu
    1 reference | 0 changes | 0 authors, 0 changes
    public bool ShowOnMenu { get; set; }

    // Danh sách các bài viết thuộc chuyên mục
    1 reference | 0 changes | 0 authors, 0 changes
    public IList<Post> Posts { get; set; }
}
```



```
// Biểu diễn tác giả của một bài viết
3 references | 0 changes | 0 authors, 0 changes
public class Author : IEntity
{
    // Mã tác giả bài viết
    2 references | 0 changes | 0 authors, 0 changes
    public int Id { get; set; }

    // Tên tác giả
    1 reference | 0 changes | 0 authors, 0 changes
    public string FullName { get; set; }

    // Tên định danh dùng để tạo URL
    1 reference | 0 changes | 0 authors, 0 changes
    public string UrlSlug { get; set; }

    // Đường dẫn tới file hình ảnh
    1 reference | 0 changes | 0 authors, 0 changes
    public string ImageUrl { get; set; }

    // Ngày bắt đầu
    1 reference | 0 changes | 0 authors, 0 changes
    public DateTime JoinedDate { get; set; }

    // Địa chỉ email
    1 reference | 0 changes | 0 authors, 0 changes
    public string Email { get; set; }

    // Ghi chú
    1 reference | 0 changes | 0 authors, 0 changes
    public string Notes { get; set; }


    // Danh sách các bài viết của tác giả
    1 reference | 0 changes | 0 authors, 0 changes
    public IList<Post> Posts { get; set; }
}
```

# Configure Model

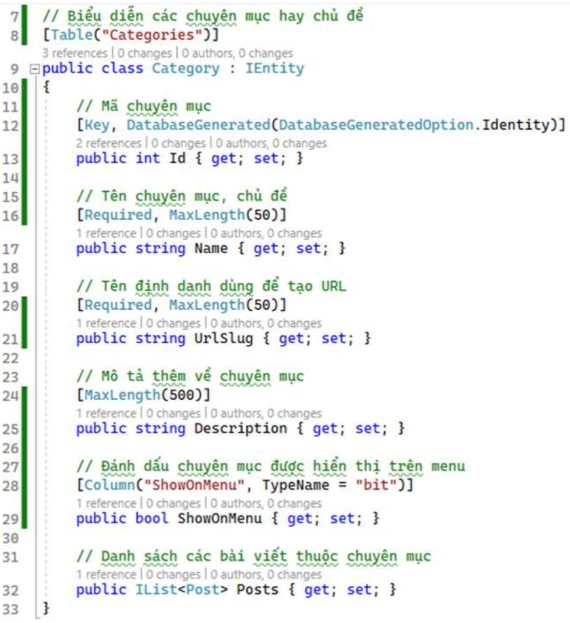
- Use set of built-in conventions
  - Table name: The name of a DbSet<T> property in the DbContext class.
  - Column name: The name of property in the entity model class.
  - The string .NET type is assumed to be a nvarchar type in the database.
  - The int .NET type is assumed to be an int type in the database.
  - The primary key is assumed to be a property that is named Id or ID, or combined entity name and Id.
  - If this property is an integer type or the Guid type, then it is also assumed to be an IDENTITY column.



```
// Biểu diễn các chuyên mục hay chủ đề
3 references | 0 changes | 0 authors, 0 changes
public class Category : IEntity
{
    // Mã chuyên mục
    2 references | 0 changes | 0 authors, 0 changes
    public int Id { get; set; }

    // Tên chuyên mục, chủ đề
    1 reference | 0 changes | 0 authors, 0 changes
    public string Name { get; set; }

    // Tên định danh dùng để tạo URL
    1 reference | 0 changes | 0 authors, 0 changes
    public string UrlSlug { get; set; }

    // Mô tả thêm về chuyên mục
    1 reference | 0 changes | 0 authors, 0 changes
    public string Description { get; set; }

    // Đánh dấu chuyên mục được hiển thị trên menu
    1 reference | 0 changes | 0 authors, 0 changes
    public bool ShowOnMenu { get; set; }

    // Danh sách các bài viết thuộc chuyên mục
    1 reference | 0 changes | 0 authors, 0 changes
    public IList<Post> Posts { get; set; }
}
```

## Configure Model

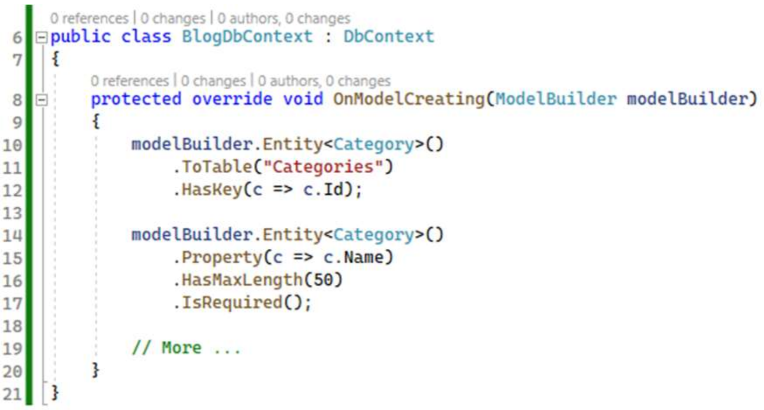
- Use data annotation attributes (mapping attributes)
  - Table(“TableName”)
  - Column(“ColumnName”)
  - Key
  - DatabaseGenerated
  - Required
  - MaxLength(50)
  - Unicode(false)
  - Precision(14, 2)
  - NotMapped
  - Comment



```
// Biểu diễn các chuyên mục hay chủ đề
[Table("Categories")]
3 references | 0 changes | 0 authors, 0 changes
public class Category : IEntity
{
    // Mã chuyên mục
    [Key, DatabaseGenerated(DatabaseGeneratedOption.Identity)]
    2 references | 0 changes | 0 authors, 0 changes
    public int Id { get; set; }

    // Tên chuyên mục, chủ đề
    [Required, MaxLength(50)]
    1 reference | 0 changes | 0 authors, 0 changes
    public string Name { get; set; }

    // Tên định danh dùng để tạo URL
    [Required, MaxLength(50)]
    1 reference | 0 changes | 0 authors, 0 changes
    public string UrlSlug { get; set; }

    // Mô tả thêm về chuyên mục
    [MaxLength(500)]
    1 reference | 0 changes | 0 authors, 0 changes
    public string Description { get; set; }

    // Đánh dấu chuyên mục được hiển thị trên menu
    [Column("ShowOnMenu", TypeName = "bit")]
    1 reference | 0 changes | 0 authors, 0 changes
    public bool ShowOnMenu { get; set; }

    // Danh sách các bài viết thuộc chuyên mục
    1 reference | 0 changes | 0 authors, 0 changes
    public IList<Post> Posts { get; set; }
}
```

## Configure Model

- Use EF Core Fluent API
  - Override the OnModelCreating method in the derived context

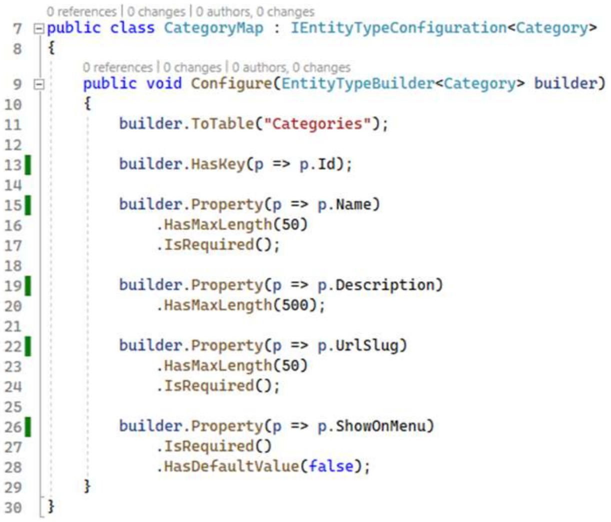


```
0 references | 0 changes | 0 authors, 0 changes
public class BlogDbContext : DbContext
{
    0 references | 0 changes | 0 authors, 0 changes
    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<Category>()
            .ToTable("Categories")
            .HasKey(c => c.Id);

        modelBuilder.Entity<Category>()
            .Property(c => c.Name)
            .HasMaxLength(50)
            .IsRequired();

        // More ...
    }
}
```

# Configure Model

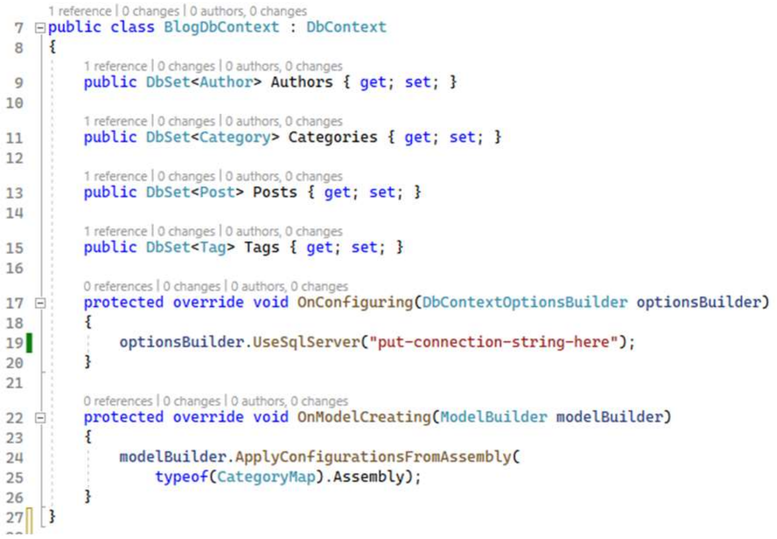
- Use grouping configuration
  - Reduce the size of the OnModelCreating method
  - Separation of concerns
  - Possible to apply all configuration specified in types implementing IEntityTypeConfiguration in a given assembly.

```
modelBuilder.ApplyConfigurationsFromAssembly(
    typeof(CategoryMap).Assembly);
```



```
0 references | 0 changes | 0 authors, 0 changes
public class CategoryMap : IEntityTypeConfiguration<Category>
{
    0 references | 0 changes | 0 authors, 0 changes
    public void Configure(EntityTypeBuilder<Category> builder)
    {
        builder.ToTable("Categories");

        builder.HasKey(p => p.Id);

        builder.Property(p => p.Name)
            .HasMaxLength(50)
            .IsRequired();

        builder.Property(p => p.Description)
            .HasMaxLength(500);

        builder.Property(p => p.UrlSlug)
            .HasMaxLength(50)
            .IsRequired();

        builder.Property(p => p.ShowOnMenu)
            .IsRequired()
            .HasDefaultValue(false);
    }
}
```

# Define Context Class



```
1 reference | 0 changes | 0 authors, 0 changes
public class BlogDbContext : DbContext
{
    1 reference | 0 changes | 0 authors, 0 changes
    public DbSet<Author> Authors { get; set; }

    1 reference | 0 changes | 0 authors, 0 changes
    public DbSet<Category> Categories { get; set; }

    1 reference | 0 changes | 0 authors, 0 changes
    public DbSet<Post> Posts { get; set; }

    1 reference | 0 changes | 0 authors, 0 changes
    public DbSet<Tag> Tags { get; set; }

    0 references | 0 changes | 0 authors, 0 changes
    protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
    {
        optionsBuilder.UseSqlServer("put-connection-string-here");
    }

    0 references | 0 changes | 0 authors, 0 changes
    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.ApplyConfigurationsFromAssembly(
            typeof(CategoryMap).Assembly);
    }
}
```

ITDLU

17

# Configure DB Provider & Connection String

- SQL Server

```
optionsBuilder.UseSqlServer(@"Server=(localdb)\\mssqllocaldb;Database=TatBlog;Trusted_Connection=True;TrustServerCertificate=True;MultipleActiveResultSets=true");
```

- SQLite

```
optionsBuilder.UseSqlite($"Data Source=D:\\path\\to\\database.db");
```

- In Memory

```
optionsBuilder.UseInMemoryDatabase(databaseName: "AuthorDb");
```

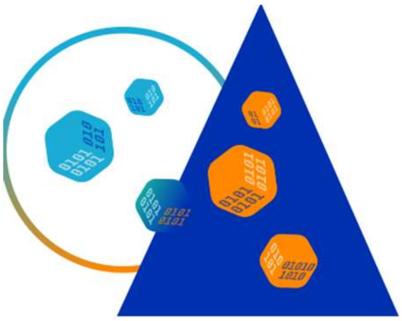
17

ITDLU

18

# Data Seeding

- Data seeding is the process of populating a database with an initial set of data.
- There are several ways this can be accomplished in EF Core:
  - Model seed data
  - Manual migration customization
  - Custom initialization logic

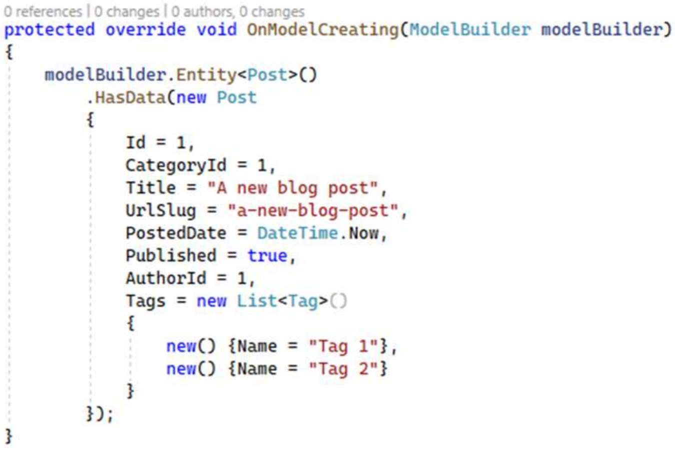


18

19

# Data Seeding

- Model seed data: Configure seed data in the method OnModelCreating
- EF Core migrations can automatically compute what insert, update or delete operations need to be applied when upgrading the database to a new version of the model.



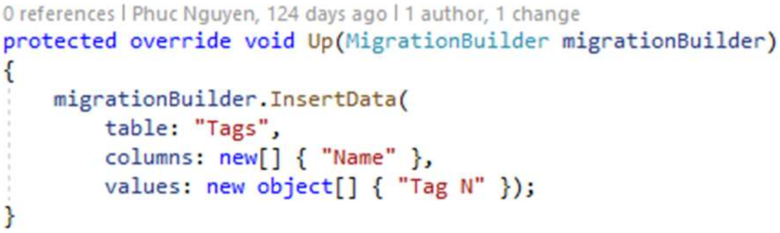
```
0 references | 0 changes | 0 authors, 0 changes
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<Post>()
        .HasData(new Post
        {
            Id = 1,
            CategoryId = 1,
            Title = "A new blog post",
            UrlSlug = "a-new-blog-post",
            PostedDate = DateTime.Now,
            Published = true,
            AuthorId = 1,
            Tags = new List<Tag>()
            {
                new() {Name = "Tag 1"},
                new() {Name = "Tag 2"}
            }
        });
}
```

19

20

# Data Seeding

- Manual migration customization:
  - Manually add the call to InsertData(), UpdateData(), and DeleteData() methods in the Migration class
  - Add custom operations to the migration



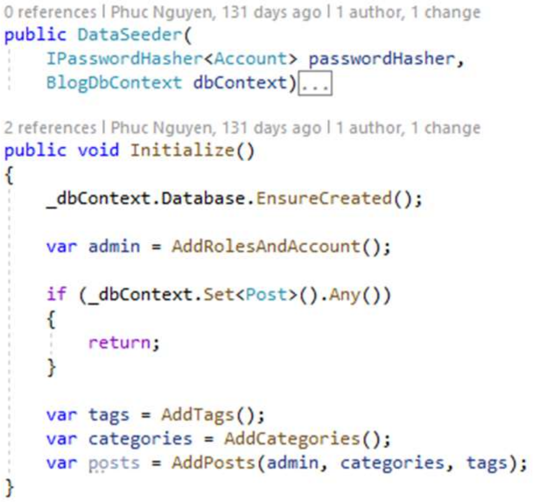
```
0 references | Phuc Nguyen, 124 days ago | 1 author, 1 change
protected override void Up(MigrationBuilder migrationBuilder)
{
    migrationBuilder.InsertData(
        table: "Tags",
        columns: new[] { "Name" },
        values: new object[] { "Tag N" });
}
```

20

21

# Data Seeding

- Custom initialization logic
  - A straightforward and powerful way
  - Use the method DbContext.SaveChanges() before the main application logic begins execution.
- Should not be part of the normal app execution as this can cause concurrency issues

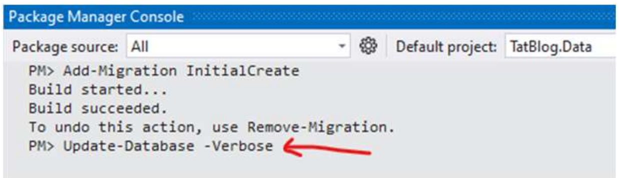


```
0 references | Phuc Nguyen, 131 days ago | 1 author, 1 change
public DataSeeder(
    IPasswordHasher<Account> passwordHasher,
    BlogDbContext dbContext)...

2 references | Phuc Nguyen, 131 days ago | 1 author, 1 change
public void Initialize()
{
    _dbContext.Database.EnsureCreated();

    var admin = AddRolesAndAccount();

    if (_dbContext.Set<Post>().Any())
    {
        return;
    }

    var tags = AddTags();
    var categories = AddCategories();
    var posts = AddPosts(admin, categories, tags);
}
```

22

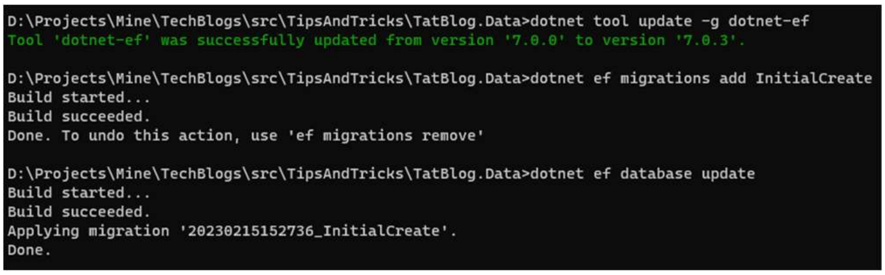
# Create Database Using EF Core Migration

- Use Package Manager Console



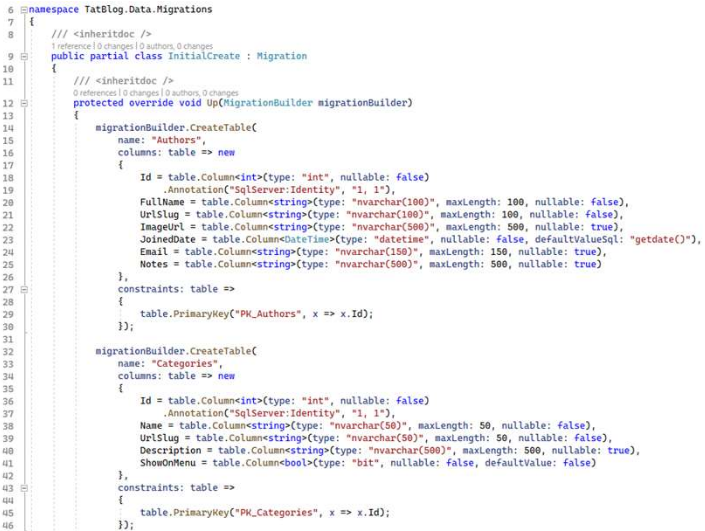
```
PM> Add-Migration InitialCreate
Build started...
Build succeeded.
To undo this action, use Remove-Migration.
PM> Update-Database -Verbose
```

- Use .NET CLI



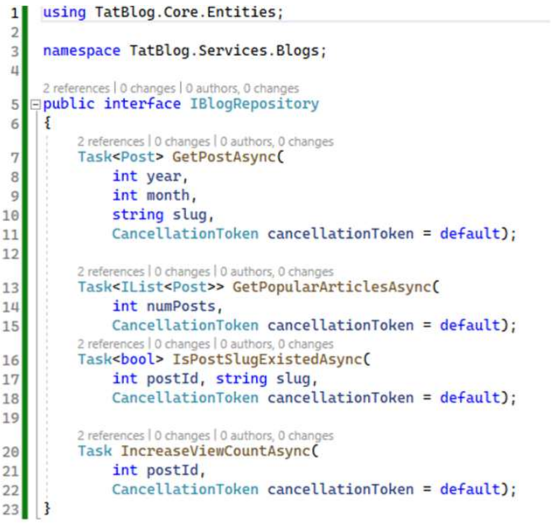
```
D:\Projects\Mine\TechBlogs\src\TipsAndTricks\TatBlog.Data>dotnet tool update -g dotnet-ef
Tool 'dotnet-ef' was successfully updated from version '7.0.0' to version '7.0.3'.

D:\Projects\Mine\TechBlogs\src\TipsAndTricks\TatBlog.Data>dotnet ef migrations add InitialCreate
Build started...
Build succeeded.
Done. To undo this action, use 'ef migrations remove'

D:\Projects\Mine\TechBlogs\src\TipsAndTricks\TatBlog.Data>dotnet ef database update
Build started...
Build succeeded.
Applying migration '20230215152736_InitialCreate'.
Done.
```

# Create Database Using EF Core Migration

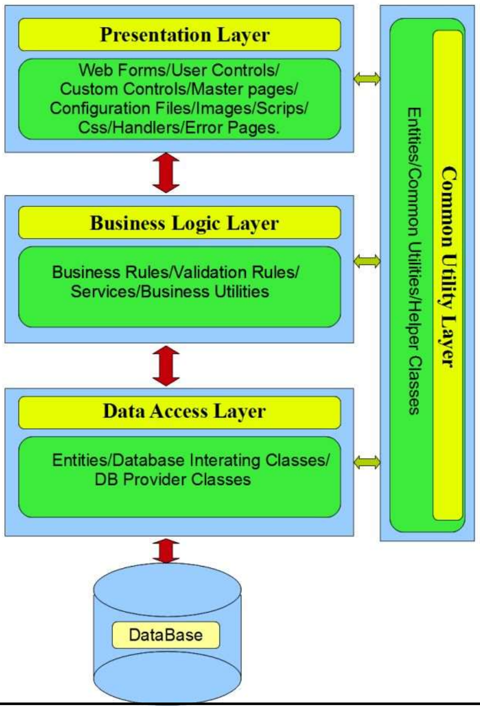


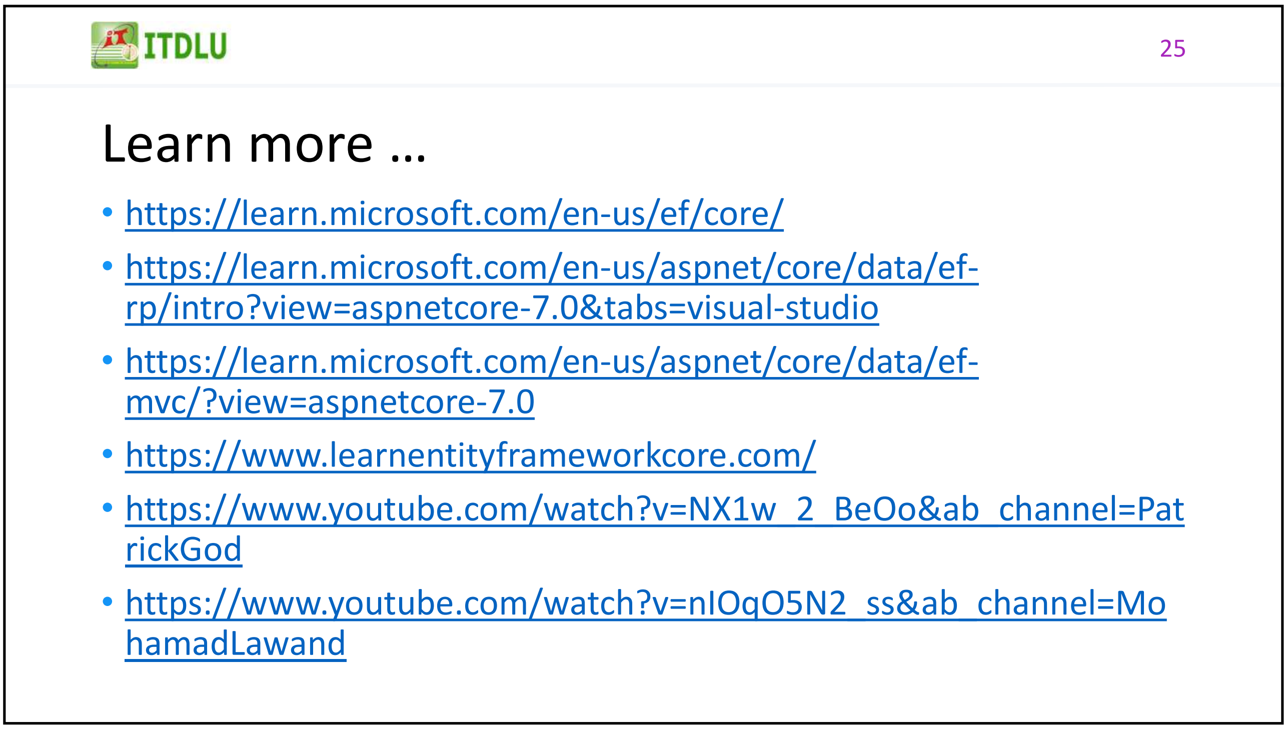
```
namespace TatBlog.Data.Migrations
{
    /// <inheritdoc />
    1 reference | 0 changes | 0 authors, 0 changes
    public partial class InitialCreate : Migration
    {
        /// <inheritdoc />
        0 references | 0 changes | 0 authors, 0 changes
        protected override void Up(MigrationBuilder migrationBuilder)
        {
            migrationBuilder.CreateTable(
                name: "Authors",
                columns: table => new
                {
                    Id = table.Column<int>(type: "int", nullable: false)
                        .Annotation("SqlServer:Identity", "1, 1"),
                    FullName = table.Column<string>(type: "nvarchar(100)", maxLength: 100, nullable: false),
                    UrlSlug = table.Column<string>(type: "nvarchar(100)", maxLength: 100, nullable: false),
                    ImageUrl = table.Column<string>(type: "nvarchar(500)", maxLength: 500, nullable: true),
                    JoinedDate = table.Column<DateTime>(type: "datetime", nullable: false, defaultValueSql: "getdate()"),
                    Email = table.Column<string>(type: "nvarchar(150)", maxLength: 150, nullable: true),
                    Notes = table.Column<string>(type: "nvarchar(500)", maxLength: 500, nullable: true)
                },
                constraints: table =>
                {
                    table.PrimaryKey("PK_Authors", x => x.Id);
                });

            migrationBuilder.CreateTable(
                name: "Categories",
                columns: table => new
                {
                    Id = table.Column<int>(type: "int", nullable: false)
                        .Annotation("SqlServer:Identity", "1, 1"),
                    Name = table.Column<string>(type: "nvarchar(50)", maxLength: 50, nullable: false),
                    UrlSlug = table.Column<string>(type: "nvarchar(50)", maxLength: 50, nullable: false),
                    Description = table.Column<string>(type: "nvarchar(500)", maxLength: 500, nullable: true),
                    ShowOnMenu = table.Column<bool>(type: "bit", nullable: false, defaultValue: false)
                },
                constraints: table =>
                {
                    table.PrimaryKey("PK_Categories", x => x.Id);
                });
```

# Build Business Logic Layer



```
using TatBlog.Core.Entities;

namespace TatBlog.Services.Blogs;

2 references | 0 changes | 0 authors, 0 changes
public interface IBlogRepository
{
    2 references | 0 changes | 0 authors, 0 changes
    Task<Post> GetPostAsync(
        int year,
        int month,
        string slug,
        CancellationToken cancellationToken = default);

    2 references | 0 changes | 0 authors, 0 changes
    Task<IList<Post>> GetPopularArticlesAsync(
        int numPosts,
        CancellationToken cancellationToken = default);
    2 references | 0 changes | 0 authors, 0 changes
    Task<bool> IsPostSlugExistedAsync(
        int postId, string slug,
        CancellationToken cancellationToken = default);

    2 references | 0 changes | 0 authors, 0 changes
    Task IncreaseViewCountAsync(
        int postId,
        CancellationToken cancellationToken = default);
}
```


Presentation Layer
Web Forms/User Controls/ Custom Controls/Master pages/ Configuration Files/Images/Scrips/ Css/Handlers/Error Pages.
Business Logic Layer
Business Rules/Validation Rules/ Services/Business Utilities
Data Access Layer
Entities/Database Interating Classes/ DB Provider Classes
DataBase
Common Utility Layer
Entities/Common Utilities/Helper Classes



## Learn more …

- https://learn.microsoft.com/en-us/ef/core/
- https://learn.microsoft.com/en-us/aspnet/core/data/ef-rp/intro?view=aspnetcore-7.0&tabs=visual-studio
- https://learn.microsoft.com/en-us/aspnet/core/data/ef-mvc/?view=aspnetcore-7.0
- https://www.learnentityframeworkcore.com/
- https://www.youtube.com/watch?v=NX1w_2_BeOo&ab_channel=PatrickGod
- https://www.youtube.com/watch?v=nIOqO5N2_ss&ab_channel=MohamadLawand

# END