**Entity Framework Core**

**[Arabic] Entity Framework Core -11 Change Table Name**

We use dataAnotation or fluintAPI for **Table Name**

STEP 1: DATAANOTATION for **Table Name**

We use Above Class for **Table Name**

[Table("Posts")]

public class Post

{

public int PostId { get; set; }

public string Title { get; set; }

public string Content { get; set; }

public Blog Blog { get; set; }

}

add-migration \_TableName

STEP 1: fluintAPI for **Table Name**

We use Class in AppDBContext for **Table Name**

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

modelBuilder.Entity<AuditEntry>().ToTable("AuditEntries");

modelBuilder.Entity<Post>().ToTable("Posts");

new BlogEntityTypeConfiguration().Configure(modelBuilder.Entity<Blog>());

//modelBuilder.Ignore<Post>();

}

add-migration \_TableName

**[Arabic] Entity Framework Core - 12 Change Schema & Map Model To View**

We use dataAnotation or fluintAPI for **Schema**

STEP 1: DATAANOTATION for **Schema**

We use Above Class

[Table("Posts",Schema="Blogging")]

public class Post

{

public int PostId { get; set; }

public string Title { get; set; }

public string Content { get; set; }

public Blog Blog { get; set; }

}

add-migration \_TableName

STEP 1: fluintAPI for **Schema**

We use Class in AppDBContext for **Schema**

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

modelBuilder.Entity<AuditEntry>().ToTable("AuditEntries");

modelBuilder.Entity<Post>().ToTable("Posts",schema: "Blogging");//Table

modelBuilder.Entity<Post>().ToView("Posts",schema: "Blogging"); //View

new BlogEntityTypeConfiguration().Configure(modelBuilder.Entity<Blog>());

//modelBuilder.Ignore<Post>();

modelBuilder.HasDefaultSchema("Blogging");//Default Schema

}

add-migration \_TableName

**[Arabic] Entity Framework Core -13 Exclude properties**

We use dataAnotation or fluintAPI for **Exclude properties**

STEP 1: DATAANOTATION for **Exclude properties**

We use Above Class for **Exclude properties**

public class Blog

{

public int BlogId { get; set; }

public string Url { get; set; }

[NotMapped]

public DateTime AddedOn { get; set; }

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

}

add-migration \_AddedOn

STEP 2: fluintAPI for **Exclude properties**

We use Class in AppDBContext for **Exclude properties**

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

//modelBuilder.Entity<AuditEntry>().ToTable("AuditEntries");

// modelBuilder.Entity<Post>().ToView("Posts",schema: "Blogging"); //View

new BlogEntityTypeConfiguration().Configure(modelBuilder.Entity<Blog>());

//modelBuilder.Ignore<Post>();

modelBuilder.Entity<Blog>().Ignore(d=>d.AddedOn); // ignore property

modelBuilder.HasDefaultSchema("Blogging");//Default Schema

}

add-migration \_AddedOn

**[Arabic] Entity Framework Core -14 Change Column Name**

We use dataAnotation or fluintAPI for **Exclude properties**

STEP 1: DATAANOTATION for **Exclude properties**

We use Above Class for **Exclude properties**

public class Blog

{

public int BlogId { get; set; }

[Column("BlogUrl")]//Column

public string Url { get; set; }

public DateTime AddedOn { get; set; }

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

}

add-migration BlogUrlCol

STEP 2: fluintAPI for **Exclude properties**

We use Class in AppDBContext for **Exclude properties**

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

modelBuilder.Entity<Blog>().Property(n=>n.Url).HasColumnName("BlogUrl"); // Rename property

}

add-migration BlogUrlCol

**[Arabic] Entity Framework Core -15 Column Data Types**

We use dataAnotation or fluintAPI for **Column Data Types**

STEP 1: DATAANOTATION for **Column Data Types**

We use Above Class for **Column Data Types**

public class Blog

{

public int BlogId { get; set; }

[Column(TypeName ="nvarchar(50)")]//Column

public string Url { get; set; }

// [NotMapped]

public DateTime AddedOn { get; set; }

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

}

add-migration TypeNameCol

STEP 2: fluintAPI for **Column Data Types**

We use Class in AppDBContext for **Column Data Types**

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

modelBuilder.Entity<Blog>().Property(n=>n.Url).HasColumnType("nvarchar(50)"); // Column Type property

modelBuilder.Entity<Blog>().Property(n=>n.Rating).HasColumnType("decimal(5,2)"); // Column Type property

}

add-migration TypeNameCol

**[Arabic[ Entity Framework Core -16 Maximum Length**

We use dataAnotation or fluintAPI for **Maximum Length**

STEP 1: DATAANOTATION for **Maximum Length**

We use Above Class for **Maximum Length**

public class Blog

{

public int BlogId { get; set; }

[MaxLength(150)]//Column

public string Url { get; set; }

// [NotMapped]

public DateTime AddedOn { get; set; }

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

}

add-migration MaxLengthCol

STEP 2: fluintAPI for **Maximum Length**

We use Class in AppDBContext for **Maximum Length**

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

modelBuilder.Entity<Blog>().Property(n=>n.Url).HasMaxLength(150); // Column MaxLength property

}

add-migration MaxLengthCol

**[Arabic[ Entity Framework Core -17 Column Comments**

We use dataAnotation or fluintAPI for **Column Comments**

STEP 1: DATAANOTATION for **Column Comments**

We use Above Class for **Column Comments**

public class Blog

{

public int BlogId { get; set; }

[Comment("Column Comment")]//Column Comment

public string Url { get; set; }

public DateTime AddedOn { get; set; }

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

}

add-migration CommentCol

STEP 2: fluintAPI for **Column Comments**

We use Class in AppDBContext for **Column Comments**

protected override void OnModelCreating(ModelBuilder modelBuilder)

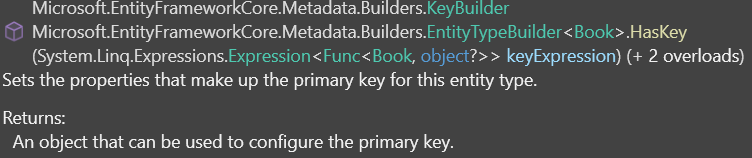
{

modelBuilder.Entity<Blog>().Property(n=>n.Url).HasComment("Column Comments"); // Column Comment property

}

add-migration CommentCol

**[Arabic] Entity Framework Core -18 Set Primary Key**

****

We use dataAnotation or fluintAPI for **Set Primary Key**

STEP 1: DATAANOTATION for **Set Primary Key**

We use Above Class for **Set Primary Key**

public class Book

{

[Key]

public int BookKey { get; set; }

public string AutherName { get; set; }

}

add-migration SetPrimaryKeyCol

STEP 2: fluintAPI for **Set Primary Key**

We use Class in AppDBContext for **Set Primary Key**

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

modelBuilder.Entity<Book>().HasKey(n=>n.BookKey); // Column Comment property

}

add-migration SetPrimaryKeyCol

**[Arabic] Entity Framework Core -19 Change Primary Key Name**

STEP 1: fluintAPI for **Change Primary Key Name**

We use Class in AppDBContext for **Change Primary Key Name**

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

modelBuilder.Entity<Book>().HasKey(n=>n.BookKey).HasName("PK\_BookKey"); // Column Comment property

}

add-migration SetPrimaryKeyCol

**[Arabic] Entity Framework Core - 20 Set Composite Key**

this is foriegn key

STEP 1: fluintAPI for **Change Primary Key Name**

We use Class in AppDBContext for **Change Primary Key Name**

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

modelBuilder.Entity<Book>().HasKey(n=>new { n.Name ,n.Author}).HasName("compiost\_Book"); // Column Comment property

}

add-migration Set\*\*Composite\*\*KeyCol

**[Arabic] Entity Framework Core - 21 Set Default Value**

STEP 2: fluintAPI for **Set Default Value**

We use Class in AppDBContext for **Set Default Value**

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

modelBuilder.Entity<Blog>().Property(b => b.rating).HasDefaultValue(2);

modelBuilder.Entity<Blog>().Property(b => b.CreateOn).HasDefaultValueSql("getdate()-1");

}

add-migration defualtvalueKeyCol

**[Arabic] Entity Framework Core - 22 Computed Columns**

STEP 2: fluintAPI for **Computed Columns**

We use Class in AppDBContext for **Computed Columns**

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

modelBuilder.Entity<Author>().Property(d => d.DisplayName).HasComputedColumnSql("[FirstName] +' ' + [LastName]");

}

add-migration ComputedvalueCol

**[Arabic] Entity Framework Core - 23 Primary Key Default Value**

We use dataAnotation or fluintAPI for **Set Primary Key**

STEP 1: DATAANOTATION for **Set Primary Key**

We use Above Class for **Set Primary Key**

public class Category

{

[DatabaseGenerated(DatabaseGeneratedOption.Identity)]

public byte Id { get; set; }

[MaxLength(50)]

public string Name { get; set; }

}

add-migration CategoriesIdentity

STEP 2: fluintAPI for **Computed Columns**

We use Class in AppDBContext for **Computed Columns**

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

modelBuilder.Entity<Category>().Property(c => c.Id).ValueGeneratedOnAdd();

}

add-migration CategoriesIdentity

**[Arabic] Entity Framework Core - 24 One To One Relationship**

One-to-one relationships are used when one entity is associated with at most one other entity. For example, a Blog has one BlogHeader, and that BlogHeader belongs to a single Blog.

We use dataAnotation or fluintAPI for **One To One Relationship**

STEP 1: DATAANOTATION for **One To One Relationship**

We use Above Class for **One To One Relationship**

// Principal (parent)

public class Blog

{

public int Id { get; set; }

public BlogHeader? Header { get; set; } // Reference navigation to dependent

}

// Dependent (child)

public class BlogHeader

{

public int Id { get; set; }

public int BlogId { get; set; } // Required foreign key property

public Blog Blog { get; set; } = null!; // Required reference navigation to principal

}

4️⃣ إدراج بيانات لاختبار العلاقة

using (var context = new ApplicationDbContext(options))

{

var user = new User

{

Name = "Ahmed",

Profile = new UserProfile

{

Address = "Riyadh, Saudi Arabia",

Phone = "0555555555"

}

};

context.Users.Add(user);

context.SaveChanges();

}

5️⃣ استرجاع بيانات المستخدم مع ملفه الشخصي

csharp

نسخ

تحرير

var userWithProfile = context.Users

.Include(u => u.Profile)

.FirstOrDefault(u => u.Id == 1);

Console.WriteLine($"User: {userWithProfile.Name}, Address: {userWithProfile.Profile.Address}");

ADD-MIGRATION ONETOONE

STEP 2: fluintAPI for **Computed Columns**

We use Class in AppDBContext for **Computed Columns**

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

modelBuilder.Entity<Blog>()

.HasOne(B => B.Header)

.WithOne(H => H.Blog)

.HasForeignKey<BlogHeader>(H => H.BlogId);

}

ADD-MIGRATION ONETOONE

**[Arabic] Entity Framework Core - 25 One To Many Relationship - Part 1**

One-to-many relationships are used when a single entity is associated with any number of other entities. For example, a Blog can have many associated Posts, but each Post is associated with only one Blog.

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

//One To Many

modelBuilder.Entity<ManyBlog>()

.HasMany(p => p.ManyPosts)

.WithOne(b => b.ManyBlog)

.HasForeignKey(e=> e.BlogId)

.HasConstraintName("FR\_ManyBlog\_ManyPosts");

modelBuilder.Entity<Company>()

.HasMany(e => e.Employees)

.WithOne(c => c.Company)

.HasForeignKey(e => e.CompanyId);

}

// Principal (parent)

public class ManyBlog

{

public int Id { get; set; }

public ICollection<ManyPost> ManyPosts { get; } = new List<ManyPost>(); // Collection navigation containing dependents

}

// Dependent (child)

public class ManyPost

{

public int Id { get; set; }

public int BlogId { get; set; } // Required foreign key property

public ManyBlog ManyBlog { get; set; } = null!; // Required reference navigation to principal

}

public class Company

{

public int Id { get; set; }

public string Name { get; set; }

// علاقة واحد إلى متعدد مع Employee

public ICollection<Employee> Employees { get; } = new List<Employee>();

}

public class Employee

{

public int Id { get; set; }

public string FullName { get; set; }

public string Position { get; set; }

// FK وعلاقته بـ Company

public int CompanyId { get; set; }

public Company Company { get; set; }

}

// in Program.cs

4️⃣ إضافة بيانات لاختبار العلاقة

csharp

نسخ

تحرير

using (var context = new ApplicationDbContext(options))

{

var company = new Company

{

Name = "Tech Solutions",

Employees = new List<Employee>

{

new Employee { FullName = "Ali Ahmed", Position = "Software Engineer" },

new Employee { FullName = "Sara Khalid", Position = "Project Manager" }

}

};

context.Companies.Add(company);

context.SaveChanges();

}

5️⃣ استرجاع بيانات الشركة مع موظفيها

csharp

نسخ

تحرير

var companyWithEmployees = context.Companies

.Include(c => c.Employees)

.FirstOrDefault(c => c.Id == 1);

Console.WriteLine($"Company: {companyWithEmployees.Name}");

foreach (var emp in companyWithEmployees.Employees)

{

Console.WriteLine($"Employee: {emp.FullName}, Position: {emp.Position}");

}

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

//One To Many

modelBuilder.Entity<Company>()

.HasMany(e => e.Employees)

.WithOne(c => c.Company)

.HasForeignKey(e => e.CompanyId);

}

ADD-MIGRATION ManyToOne

public class CompanyTest

{

public int Id { get; set; }

public string Name { get; set; }

public int TestId { get; set; }

public int TestName { get; set; }

// علاقة واحد إلى متعدد مع Employee

public ICollection<EmployeeTest> EmployeesTest { get; } = new List<EmployeeTest>();

}

public class EmployeeTest

{

public int Id { get; set; }

public int TestId { get; set; }

public int TestName { get; set; }

public string FullName { get; set; }

public string Position { get; set; }

// FK وعلاقته بـ Company

public CompanyTest CompanyTest { get; set; }

}

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

//One To Many CompositKey

modelBuilder.Entity<CompanyTest>()

.HasMany(e => e.EmployeesTest)

.WithOne(c => c.CompanyTest)

.HasForeignKey(e => new { e.TestId, e.TestName })

.HasPrincipalKey(c => new { c.TestId, c.TestName });

}

**[Arabic] Entity Framework Core - 26 One To Many Relationship - Part 2**

// إضافة بيانات

var company = new CompanyTest

{

Id = 1,

Name = "Tech Corp",

TestId = 101,

TestName = 202,

EmployeesTest = new List<EmployeeTest>

{

new EmployeeTest { Id = 1, TestId = 101, TestName = 202, FullName = "Ali Ahmed", Position = "Software Engineer", CompanyTestId = 1 },

new EmployeeTest { Id = 2, TestId = 101, TestName = 202, FullName = "Sara Khalid", Position = "Project Manager", CompanyTestId = 1 }

}

};

context.CompaniesTest.Add(company);

context.SaveChanges();

Console.WriteLine("Data Inserted Successfully!");

// استرجاع بيانات الشركة مع موظفيها

var companyWithEmployees = context.CompaniesTest

.Include(c => c.EmployeesTest)

.FirstOrDefault(c => c.Id == 1 && c.TestId == 101 && c.TestName == 202);

if (companyWithEmployees != null)

{

Console.WriteLine($"Company: {companyWithEmployees.Name}");

foreach (var emp in companyWithEmployees.EmployeesTest)

{

Console.WriteLine($"Employee: {emp.FullName}, Position: {emp.Position}");

}

}

else

{

Console.WriteLine("Company not found.");

}

ADD-MIGRATION ManyToOne

**[Arabic] Entity Framework Core - 27 Many To Many Relationship**

Many-to-many relationships are used when any number entities of one entity type is associated with any number of entities of the same or another entity type. For example, a Post can have many associated Tags, and each Tag can in turn be associated with any number of Posts.

public class Post

{

public int Id { get; set; }

public List<PostTag> PostTags { get; } = [];

}

public class Tag

{

public int Id { get; set; }

public List<PostTag> PostTags { get; } = [];

}

public class PostTag

{

public int PostsId { get; set; }

public int TagsId { get; set; }

public Post Post { get; set; } = null!;

public Tag Tag { get; set; } = null!;

}

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

//Many To Many

modelBuilder.Entity<PostTag>(entity =>

{

// تعيين المفتاح المركب باستخدام TagId و PostId

entity.HasKey(pt => new { pt.TagsId, pt.PostsId });

// تعريف العلاقة بين Post و PostTag

entity.HasOne(pt => pt.TestPost)

.WithMany(p => p.PostTags)

.HasForeignKey(pt => pt.PostsId);

// تعريف العلاقة بين Tag و PostTag

entity.HasOne(pt => pt.Tag)

.WithMany(t => t.PostTags)

.HasForeignKey(pt => pt.TagsId);

});

}

using Microsoft.EntityFrameworkCore;

using System;

using System.Collections.Generic;

using System.Linq;

class Program

{

static void Main()

{

var \_context = new ApplictionDBContext();

// إنشاء منشور جديد

var testPost= new TestPost();

// إنشاء وسوم جديدة

var tag1 = new Tag();

var tag2 = new Tag();

// حفظ البيانات الأساسية

context.TestPosts.Add(testPost);

context.Tags.AddRange(tag1, tag2);

context.SaveChanges();

// ربط المنشور بالوسوم عبر جدول PostTag

var postTag1 = new PostTag { PostId = post.Id, TagId = tag1.Id };

var postTag2 = new PostTag { PostId = post.Id, TagId = tag2.Id };

context.PostTags.AddRange(postTag1, postTag2);

context.SaveChanges();

Console.WriteLine("✅ البيانات أُدرجت بنجاح!");

}

}

using (var context = new ApplicationDbContext(new DbContextOptionsBuilder<ApplicationDbContext>()

.UseSqlServer("Server=.;Database=TestDb;Trusted\_Connection=True;TrustServerCertificate=True;")

.Options))

{

// استرجاع جميع المنشورات مع الوسوم المرتبطة بها

var postsWithTags = context.Posts

.Include(p => p.PostTags)

.ThenInclude(pt => pt.Tag)

.ToList();

foreach (var post in postsWithTags)

{

Console.WriteLine($"📝 المنشور ID: {post.Id}");

foreach (var postTag in post.PostTags)

{

Console.WriteLine($" 🏷 الوسم ID: {postTag.TagId}");

}

}

}

**[Arabic] Entity Framework Core - 29 Indexes**

Indexes are a common concept across many data stores. While their implementation in the data store may vary, they are used to make lookups based on a column (or set of columns) more efficient. See the [indexes section](https://learn.microsoft.com/en-us/ef/core/performance/efficient-querying#use-indexes-properly) in the performance documentation for more information on good index usage.

STEP 1: DATAANOTATION for **One To One Relationship**

We use Above Class for **One To One Relationship**

[Index(nameof(DeptID))]

public class Company

{

public int Id { get; set; }

public string Name { get; set; }

public string DeptID { get; set; }

// علاقة واحد إلى متعدد مع Employee

public ICollection<Employee> Employees { get; } = new List<Employee>();

}

STEP 2: fluintAPI for **Computed Columns**

We use Class in AppDBContext for **Computed Columns**

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

//Index

modelBuilder.Entity<Company>()

.HasIndex(c => c.DeptID);

}

**[Arabic] Entity Framework Core - 30 Composite Index**

STEP 1: DATAANOTATION for **One To One Relationship**

We use Above Class for **One To One Relationship**

[Index(nameof(FullName),nameof(Position))]

public class Employee

{

public int Id { get; set; }

public string FullName { get; set; }

public string Position { get; set; }

// FK وعلاقته بـ Company

public int CompanyId { get; set; }

public Company Company { get; set; }

}

STEP 2: fluintAPI for **Computed Columns**

We use Class in AppDBContext for **Computed Columns**

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

//Index Composit

modelBuilder.Entity<Employee>()

.HasIndex(c => new { c.FullName,c.Position });

}

**[Arabic] Entity Framework Core - 31 Index Uniqueness**

STEP 1: DATAANOTATION for **One To One Relationship**

We use Above Class for **One To One Relationship**

[Index(nameof(DeptID),IsUnique=true)]

public class Company

{

public int Id { get; set; }

public string Name { get; set; }

public string DeptID { get; set; }

// علاقة واحد إلى متعدد مع Employee

public ICollection<Employee> Employees { get; } = new List<Employee>();

}

STEP 2: fluintAPI for **Computed Columns**

We use Class in AppDBContext for **Computed Columns**

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

modelBuilder.Entity<Company>()

.HasIndex(c => c.DeptID).IsUnique();

}

**[Arabic] Entity Framework Core - 32 Index Name**

STEP 1: DATAANOTATION for **One To One Relationship**

We use Above Class for **One To One Relationship**

[Index(nameof(DeptID),Name="Index\_Test")]

public class Company

{

public int Id { get; set; }

public string Name { get; set; }

public string DeptID { get; set; }

// علاقة واحد إلى متعدد مع Employee

public ICollection<Employee> Employees { get; } = new List<Employee>();

}

STEP 2: fluintAPI for **Computed Columns**

We use Class in AppDBContext for **Computed Columns**

protected override void

OnModelCreating(ModelBuilder modelBuilder)

{

modelBuilder.Entity<Company>()

.HasIndex(c => c.DeptID)

.HasDatabaseName("Index\_Test");

}

**[Arabic] Entity Framework Core - 33 Index Filter**

this will allow null

STEP 1: fluintAPI for **Computed Columns**

We use Class in AppDBContext for **Computed Columns**

protected override void

OnModelCreating(ModelBuilder modelBuilder)

{

modelBuilder.Entity<Company>()

.HasIndex(c => c.DeptID).IsUnique()

.HasFilter(null);

}

**[Arabic] Entity Framework Core - 34 Sequences**

A sequence generates unique, sequential numeric values in the database. Sequences are not associated with a specific table, and multiple tables can be set up to draw values from the same sequence.

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

modelBuilder.HasSequence<int>("OrderNumbers");

// modelBuilder.HasSequence<int>("OrderNumbers", schema: "shared")

//.StartsAt(1000)

//.IncrementsBy(5);

modelBuilder.Entity<Order>()

.Property(o => o.OrderNo)

.HasDefaultValueSql("NEXT VALUE FOR OrderNumbers");

}

**[Arabic] Entity Framework Core - 35 Data Seeding**

حيث يتم إدخال البيانات يدويًا عند تشغيل التطبيق.

class Program

{

static void Main()

{

var \_context = new ApplictionDBContext();

// استدعاء دالة SeedData

SeedData(\_context);

// التأكد من أن البيانات تم إدراجها

RetrieveData(\_context);

}

static void SeedData(ApplictionDBContext context)

{

// التأكد من أن قاعدة البيانات موجودة

context.Database.EnsureCreated();

// التحقق مما إذا كانت البيانات قد أُدرجت مسبقًا

if (!context.TestPosts.Any() && !context.Tags.Any())

{

// إنشاء منشورات

var post1 = new TestPost { Title = "EF Core Guide" };

var post2 = new TestPost { Title = "ASP.NET Core Best Practices" };

// إنشاء وسوم

var tag1 = new Tag { Name = "C#" };

var tag2 = new Tag { Name = "Entity Framework" };

// إضافة البيانات إلى السياق

context.TestPosts.AddRange(post1, post2);

context.Tags.AddRange(tag1, tag2);

context.SaveChanges(); // حفظ البيانات الأساسية

// إنشاء العلاقات Many-to-Many

var postTag1 = new PostTag { PostsId = post1.Id, TagsId = tag1.Id };

var postTag2 = new PostTag { PostsId = post1.Id, TagsId = tag2.Id };

var postTag3 = new PostTag { PostsId = post2.Id, TagsId = tag1.Id };

context.PostTags.AddRange(postTag1, postTag2, postTag3);

context.SaveChanges(); // حفظ العلاقات

Console.WriteLine("✅ البيانات أُدرجت بنجاح!");

}

else

{

Console.WriteLine("⚠️ البيانات موجودة بالفعل، لا حاجة للإدراج.");

}

}

static void RetrieveData(ApplictionDBContext context)

{

var postsWithTags = context.TestPosts

.Include(p => p.PostTags)

.ThenInclude(pt => pt.Tag)

.ToList();

foreach (var post in postsWithTags)

{

Console.WriteLine($"📝 المنشور: {post.Title}");

foreach (var postTag in post.PostTags)

{

Console.WriteLine($" 🏷 الوسم: {postTag.Tag.Name}");

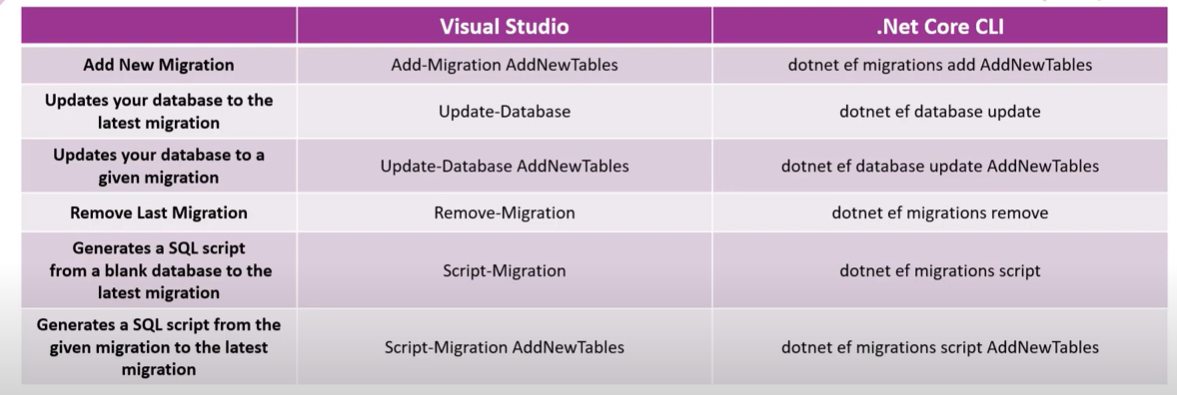
}

}

}

}

**[Arabic] Entity Framework Core - 36 Manage Migration and Generate SQL Scripts**



**[Arabic] Entity Framework Core - 37 Working With an Existing Database (Database Scaffolding)**

This is to apply Scaffolding for all database Tables

scaffold-dbContext 'server=localhost\\SQLEXPRESS; database=EFCoreDB;User Id=sa;Password=123;' Microsoft.EntityFrameworkCore.SqlServer

This is to apply Scaffolding for one by name database Tables

scaffold-dbContext 'server=localhost\\SQLEXPRESS; database=EFCoreDB;User Id=sa;Password=123;' Microsoft.EntityFrameworkCore.SqlServer -Tables Blogs

This is to apply Scaffolding for all database Tables Models Folder

scaffold-dbContext 'server=localhost\\SQLEXPRESS; database=EFCoreDB;User Id=sa;Password=123;Encrypt=false;MultipleActiveResultSets=true' Microsoft.EntityFrameworkCore.SqlServer -OutputDir

This is to apply Scaffolding for all database Tables Models Folder with Context Data Folder

scaffold-dbContext 'server=localhost\\SQLEXPRESS; database=EFCoreDB;User Id=sa;Password=123;Encrypt=false;MultipleActiveResultSets=true' Microsoft.EntityFrameworkCore.SqlServer -OutputDir -ContextDir Data

This is to apply Scaffolding for all database Tables Models Folder with Context Data Folder with rename DBContext Class

scaffold-dbContext 'server=localhost\\SQLEXPRESS;database=EFCoreDB;User Id=sa;Password=123;Encrypt=false;MultipleActiveResultSets=true' Microsoft.EntityFrameworkCore.SqlServer -OutputDir Models -ContextDir Data -Context ApplicationDBContext

This is to apply Scaffolding for all database Tables Models Folder with Context Data Folder with rename DBContext Class with **DataAnotation**

scaffold-dbContext 'server=localhost\\SQLEXPRESS;database=EFCoreDB;User Id=sa;Password=123;Encrypt=false;MultipleActiveResultSets=true' Microsoft.EntityFrameworkCore.SqlServer -OutputDir Models -ContextDir Data -Context ApplicationDBContext -DataAnnotations

**[Arabic] Entity Framework Core - 38 Select All Data, Select One Item Using .Find**

**Select All Data**

var stock = \_context.Stocks.Find(100);

Console.WriteLine($"ID:{stock.Id} , Name: {stock.FirstName}");

**Select One Item Using .Find**

var \_context = new ApplicationDBContext();

var stock = \_context.Stocks.ToList();

foreach (var St in stock)

Console.WriteLine(St.FirstName);

**[Arabic] Entity Framework Core - 39 Select One Item Using .Single**

var st = \_context.Stocks.Single(m => m.Id == 100);

Console.WriteLine($"ID:{st.Id} , Name: {st.FirstName}");

Console.ReadLine();

var stD = \_context.Stocks.SingleOrDefault(m => m.Id == 100);

Console.WriteLine($"ID:{stD.Id} , Name: {stD.FirstName}");

Console.ReadLine();

**[Arabic] Entity Framework Core - 40 Select One Item Using .First**

var stF = \_context.Stocks.First(m => m.Id > 101);

Console.WriteLine($"ID:{stF.Id} , Name: {stF.FirstName}");

Console.ReadLine();

var stFD = \_context.Stocks.First(m => m.Id > 101);

Console.WriteLine($"ID:{stFD.Id} , Name: {stFD.FirstName}");

Console.ReadLine();

**[Arabic] Entity Framework Core - 41 Select One Item Using .Last**

var stL = \_context.Stocks.OrderBy(m=>m.Id).Last();

Console.WriteLine($"ID:{stL.Id} , Name: {stL.FirstName}");

Console.ReadLine();

var stLD = \_context.Stocks.OrderBy(m => m.Id).LastOrDefault();

Console.WriteLine($"ID:{stLD.Id} , Name: {stLD.FirstName}");

Console.ReadLine();

**[Arabic] Entity Framework Core - 42 Filtering Data Using .Where**

Limits the query to results that match specified filtering criteria.

var stocks = \_context.Stocks.Where(m=>m.Id>500 && m.FirstName.StartsWith("Z")).ToList();

foreach (var St in stocks)

Console.WriteLine($"id:{St.Id},{St.FirstName}");

Console.ReadLine();

**[Arabic] Entity Framework Core - 43 .Any vs .All**

var stocks = \_context.Stocks.Any(m=>m.Id>500 );

Console.WriteLine(stocks);

Console.ReadLine();

var stocks = \_context.Stocks.All(m=>m.Id>0 );

Console.WriteLine(stocks);

Console.ReadLine();

**[Arabic] Entity Framework Core - 44 .Append vs .Prepend**

**Append**

Add last of list

var stocks = \_context.Stocks.Where(m => m.Id > 500 ).ToList().Append(new Stock { Id=1001,FirstName="Saleh"});

foreach (var St in stocks)

Console.WriteLine($"id:{St.Id},{St.FirstName}");

Console.ReadLine();

**Prepend**

Add First of list

var stocks = \_context.Stocks.Where(m => m.Id > 500 ).ToList().Prepend(new Stock { Id=500,FirstName="Saleh"});

foreach (var St in stocks)

Console.WriteLine($"id:{St.Id},{St.FirstName}");

Console.ReadLine();

**[Arabic] Entity Framework Core - 45 .Average vs .Count vs .Sum**

**Average:**

var stocks = \_context.Stocks.Average(m => m.Id);

Console.WriteLine($"id:{stocks}");

Console.ReadLine();

**Count: (LongCount)**

var stocks = \_context.Stocks.Count();

Console.WriteLine($"id:{stocks}");

Console.ReadLine();

**Sum:**

**[Arabic] Entity Framework Core - 46 .Max vs .Min**

**Max:**

var stocks = \_context.Stocks.Max(m=>m.Id);

Console.WriteLine($"id:{stocks}");

Console.ReadLine();

**Min:**

var stocks = \_context.Stocks.Min(m=>m.FirstName);

Console.WriteLine($"id:{stocks}");

Console.ReadLine();

**[Arabic] Entity Framework Core - 47 Data Sorting Using .OrderBy**

var stocks = \_context.Stocks.Where(m => m.Id > 500).ToList().OrderByDescending(m=>m.Id);

foreach (var St in stocks)

Console.WriteLine($"id:{St.Id},{St.FirstName}");

Console.ReadLine();

var stocks = \_context.Stocks.Where(m => m.Id > 500).ToList().OrderBy(m=>m.FirstName).ThenByDescending(m=>m.FirstName);

foreach (var St in stocks)

Console.WriteLine($"id:{St.Id},{St.FirstName}");

Console.ReadLine();

var stocks = \_context.Stocks.Where(m => m.Id > 500).ToList().OrderBy(m=>m.Id);

foreach (var St in stocks)

Console.WriteLine($"id:{St.Id},{St.FirstName}");

Console.ReadLine();

**[Arabic] Entity Framework Core - 48 Projection Using .Select**

var stocks = \_context.Stocks.Where(m => m.Id > 500).Select(m => new {StockId=m.Id,Name=m.FirstName }).ToList().OrderBy(m=>m.Name).ThenByDescending(m=>m.Name);

foreach (var St in stocks)

Console.WriteLine($"id:{St.StockId},{St.Name}");

Console.ReadLine();

**[Arabic] Entity Framework Core - 49 Select Unique Values Using .Distinct**

var stocksD = \_context.Stocks.Where(m => m.Id > 500).Select(m => new {Name=m.FirstName }).Distinct().ToList();

foreach (var St in stocksD)

Console.WriteLine($"id:{St.Name}");

Console.ReadLine();

**[Arabic] Entity Framework Core - 50 .Take vs .Skip**

var stockss = \_context.Stocks.Where(m => m.Id > 500).Select(m => new {m.Id }).Skip(5).Take(10).ToList();

foreach (var St in stockss)

Console.WriteLine($"id:{St.Id}");

Console.ReadLine();

**[Arabic] Entity Framework Core - 51 .GroupBy**

var stockss = \_context.Stocks.Where(m => m.Id > 0).GroupBy(m=>m.Gender).Select(m => new {Gender=m.Key,Count=m.Count() }).ToList();

foreach (var St in stockss)

Console.WriteLine($"id:{St.Count},{St.Gender}");

Console.ReadLine();