Entity Framework

حسن شرف

ايه هى ال Entity Framework

هو إطار عمل من مايكروسوفت بيسهل التعامل مع قواعد البيانات من غير ما تكتب كود SQL بنفسك. يعني تقدر تتعامل مع البيانات في الجداول على إنها كائنات (Classes) في C# والعكس (هنعرف ليه العكس متستعجلش).

الفكرة الأساسية من الـ EF هي إنه يحول الجداول اللي في قاعدة البيانات لكلاسات والعكس، علشان تقدر تعمل عمليات زي الإضافة، القراءة، التعديل، والحذف (CRUD) بسهولة.

أنواع الـ Mapping فى Entity Framework

يعنى ايه mapping الاول دى؟ العملية الى قولنا عليها فوق الى هى لو عندى class فى #c بحوله ل table فى ال database يعنى الموضوع رايح جاى ممكن اعمل ده او اعمل ده وهنا يجى الكلام على الانواع بتاعة ال mapping والحقيقة الموضوع مر بشوية مراحل كده هنعرفها مع بعض

TPC (Table Per Class)

TPH (Table Per Hierarchy)

TPCC (Table Per Concrete Class):

ايه الكلام الكبير ده انا مش فاهم حاجة تعال معايا نفهم كل مرحلة كانت ايه وليه فى كذا مرحلة

1. TPC (Table Per Class):

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

public class FullTimeEmployee : Employee
{
    public decimal Salary { get; set; }
}

public class PartTimeEmployee : Employee
{
    public decimal HourRate { get; set; }
    public int CountOfHours { get; set; }
}
```

في TPC، كل كلاس بيتم تمثيله بجدول مستقل في قاعدة البيانات. ال class الأب (base class) بيكون له جدول مستقل عن الكلاسات الموروثة منه.

- هنا كل (Class) في الكود بيتحول لجدول مستقل في قاعدة البيانات.
- مثال: لو عندك كلاس اسمه Employee فيه خصائص زي Id, Name, Age, الجدول هيكون بالشكل ده:
- جدولEmployee فيه الأعمدة: Id, Name, Age, وهكذا باقى الجداول

2. TPH (Table Per Hierarchy):

```
blic class Employee

  public int Id { get; set; }
  public string Name { get; set; }
  public int Age { get; set; }
  public decimal? Salary { get; set; }
  public decimal? HourRate { get; set; }
  public int? CountOfHours { get; set; }
  public string Discriminator { get; set; } // New column to identify the type


blic class FullTimeEmployee : Employee

  public FullTimeEmployee()
  {
      Discriminator = "FullTime";
  }


blic class PartTimeEmployee : Employee

  public PartTimeEmployee()
  {
      Discriminator = "PartTime";
  }
```

في **TPH**، كل الكلاسات الموروثة (subclasses) بتتجمع في **جدول واحد**، وبيتم إضافة عمود إضافي لتمييز النوع (discriminator column).

- هنا كل الكلاسات الموروثة (Classes Inherited) بتتجمع في جدول واحد، ويتم إضافة عمود جديد لتمييز النوع.
- مثال: عندك Employee و FullTimeEmployee و PartTimeEmployee. الجداول هتكون بالشكل ده:
- جدول واحد اسمه Employees فيه الأعمدة: Id, Name, Age, Salary, Discriminator، والـ Discriminator بيحدد نوع الموظف.

3. TPCC (Table Per Concrete Class):

```
public class FullTimeEmployee
{
    public int Id { get; set; }
    public string Name { get; set; }
    public int Age { get; set; }
    public decimal Salary { get; set; }
    public DateTime StartDate { get; set; }
}

public class PartTimeEmployee
{
    public int Id { get; set; }
    public string Name { get; set; }
    public int Age { get; set; }
    public decimal HourRate { get; set; }
    public int CountOfHours { get; set; }
}
```

في TPCC، كل كلاس فرعي (subclass) بيكون له جدول مستقل بيحتوي على الخصائص الفريدة بس. الكلاس الأب مبيكونش ليه جدول مستقل.

- هنا كل كلاس فرعي بيتحول لجدول مستقل بيحتوي على الخصائص الفريدة بس.
- مثال: عندك FullTimeEmployee و PartTimeEmployee. الجداول هتكون بالشكل ده:
- جدول FullTimeEmployees فيه الأعمدة: Id, Name, Age, Salary, StartDate.
- جدول PartTimeEmployees فيه الأعمدة: Id, Name, Age, HourRate, CountOfHours.

استخدم كل نوع امتى ؟

- **TPC مناسب لو عايز جداول مستقلة لكل كلاس، لكن مشكلته في التكرار والأداء.**
- **TPH بيسهل التجميع في جدول واحد، لكن ممكن يكون حجمه ضخم وفيه قيم null كتير.**
- **TPCC بيديك جداول مستقلة للكلاسات الفرعية، لكن ممكن يسبب تكرار وتعقيد في الاستعلامات.**

الاختيار بين الأنواع دي بيعتمد على احتياجات المشروع

ازاى بقا بحول ال classes ل tables او العكس

عشان عملية التحويل تحصل عندى طريقتين :-

اول طريقة وهى إلى بنستخدمها اغلب الوقت وهى Code First

تانى طريقة ودى بنعمل فيها الداتابيز الاول وهى Database First

خطوات العمل مع Entity Framework باستخدام Code First

- **Code First**: الطريقة دي بتبدأ بكتابة الكود اللي بيمثل الكلاسات، وبعد كده EF بيولد الجداول في قاعدة البيانات. وال class ده بيتسمى باسماء كتير زى
- (model - domain model - Entity - Poco class)

ايه الاسم ده Poco class ؟

- POCO Class هو كلاس بسيط بيحتوي على خصائص (Properties) فقط بدون أي وظائف إضافية (methods) أو خصائص متقدمة. الغرض منه هو إنه يمثل البيانات بشكل بسيط ومنظم، وغالبًا بيستخدم في الـ Entity Framework عشان يمثل الكيانات (Entities) فى قاعدة البيانات.

بعد ما جهزنا ال models ايه الى بعد كده ؟

Entity Framework بيدعم 4 طرق رئيسية لمطابقة الكلاسات مع الجداول في قاعدة البيانات.
كل طريقة منهم بتحدد إزاي الـ entities (الكلاسات) بتترجم لسكيمة قاعدة البيانات (الجداول).
وال 4 طرق دول هما :

1. **المطابقة بالاتفاقية (Convention-Based Mapping)**
2. **الـ Data Annotations**
3. **الـ Fluent API**
4. **الإعداد اليدوي (Custom Code)**

هنتكلم فى الملف ده عن اول نوع بس ونعرف الدورة كلها بتم ازاى وبعد كده هيكون فى ملف تانى لكل نوع بالتفصيل

خطوات العمل مع Entity Framework باستخدام Code First

1. المطابقة بالاتفاقية (Convention-Based Mapping)
إزاي بتشتغل:

- **اسم الكلاس واسم الجدول: الـ EF بيفترض إن كل كلاس هيتحول لجدول بنفس الاسم (بس بيجمع الاسم لو كان مفرد).**
- **اسم الخصائص واسم الأعمدة: بيفترض إن كل خاصية في الكلاس هتبقى عمود بنفس الاسم في الجدول.**
- **المفتاح الأساسي: بيفترض إن أي خاصية اسمها Id أو [Id[ClassName هي المفتاح الأساسي.**
- **العلاقات: الـ EF بيلقط العلاقات تلقائيًا بناءً على الخصائص المتنقلة (navigation properties) واتفاقيات المفاتيح الخارجية (foreign keys).**

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

النتيجة:

- هيتم إنشاء جدول اسمه Employees.
- الأعمدة هتكون Id, Name, Age.
- الـ Id هيبقى المفتاح الأساسي (Primary Key).

كل ده حصل بدون تدخل منى او اى configuration هى ال entity عملته كله لوحدها زى ما مكتوب فوق بإتفاق مسبق

هل كده ال database اتعملت يعنى لو فتحت sql server هلاقيها ؟

الإجابة لا فى خطوات تانى لسه احنا كده عملنا خطوتين بس خلينا نفتكرهم مع بعض:

1- جهزنا ال models بتاعتنا الى هتتحول ل tables

2-حددنا شكل ال tables هيكون ازاى واحد يقولى هو احنا عملنا حاجه ؟ هقولك اه احنا سيبنا ال default configuration زى ما هو يعنى كده احنا موافقين عليه

هنتكلم فى ايه بعد كده انشاء DbContext يطلع ايه ده كمان؟

إنشاء DbContext:

- هنا بنعرف كلاس جديد اسمه CompanyDbContext.
- الكلاس ده بيورث من DbContext، اللي هو الكلاس الرئيسي في Entity Framework للتعامل مع قاعدة البيانات.
- الكلاس DbContext هو الأساس في EF، وهو اللي بيوفر لك الوسيط بين الكود بتاعك وقاعدة البيانات. بمعنى تاني، هو اللي بيخليك تقدر تشتغل على البيانات كأنها كائنات (Objects) في C# بدل ما تكتب استعلامات SQL.
- يفضل يكون اسم ال class هو اسم ال database +DbContext

```
public class CompanyDbContext : DbContext
```

تعريف DbSet

هنا ببساطة بعرف كل ال classes الى عندى ك Dbset

- DbSet<Employee> هو نوع خاص من الكائنات اللي بيمثل جدول في قاعدة البيانات.

- Employees هو اسم الخاصية (Property) اللي بتستخدمها علشان تتعامل مع جدول الموظفين في قاعدة البيانات.
- الخاصية دي بتسمح لك تعمل عمليات على البيانات زي:
- الإضافة: يعني تقدر تضيف موظف جديد للجدول.
- القراءة: يعني تقدر تستخرج بيانات الموظفين من الجدول.
- التعديل: يعني تقدر تعدل بيانات موظف موجود بالفعل.
- الحذف: يعني تقدر تحذف موظف من الجدول.
- ببساطة، DbSet<Employe> بتخلي التعامل مع جدول Employees في قاعدة البيانات سهل وكأنه مجرد قائمة (List) من الموظفين في الكود.

```
public DbSet<Employee> Employees { get; set; }  // ده جدول الموظفين
```

تعريف OnConfiguring

- بتستخدم علشان تحدد إعدادات الاتصال بقاعدة البيانات
- بنعمل Override للطريقة دي علشان نحدد إعدادات خاصة بقاعدة البيانات اللي هنشتغل عليها.

```
optionsBuilder.UseSqlServer("Server=.;Database=EnterpriseGr03;Trusted_Connection=True;");
```

- optionsBuilder: ده الكائن اللي بنستخدمه علشان نبني إعدادات الاتصال بقاعدة البيانات.
- هنا بحدد السيرفر الى عليه ال database واسم ال database وال security

تعريف Migration

الـ Migration في Entity Framework هو عبارة عن آلية بتسمح لك بتتبع وتطبيق التعديلات اللي بتعملها على الكلاسز (الكائنات) الخاصة بالـ Entities بتاعتك على قاعدة البيانات. يعني لو غيرت في الكود أو أضفت خصائص جديدة، تقدر تستخدم الـ Migrations عشان تطبق التعديلات دي على قاعدة البيانات بشكل سلس.

```
Add-Migration "InitialCreate"
```

هنا انا عملت اول migration والمتعارف عليه انها يكون اسمها كده لانى لسه بنشئ ال database بس لسه بردو الكلام ده مسمعش عندى فى sql server هنا كل الى عمله عمل class

الكلاس ده بيحتوي على دالتين أساسيتين:

- **Up Method:**
- **الدالة دي بتنفذ لما عايز تطبق التغييرات على قاعدة البيانات.**
- **أي أكواد في الدالة دي هتتحول لتعليمات SQL لتعديل قاعدة البيانات، زي إنشاء جداول جديدة أو تعديل جداول موجودة.**

- **Down Method:**
- **الدالة دي هي العكس من Up.**
- **الدالة دي بتستخدم لما عايز تلغي التغييرات اللي حصلت في Up.**
- **مثلا لو الدالة Up أضافت جدول جديد، الدالة Down هتقوم بحذف الجدول ده.**

Example of up method

```
protected override void Up(MigrationBuilder migrationBuilder)
                                                        {
                                        migrationBuilder.CreateTable(
                                                    name: "Employees",
                                                columns: table => new
                                                        {
                                            Id = table.Column<int>(nullable: false)
                                        .Annotation("SqlServer:Identity", "1, 1"),
                                        Name = table.Column<string>(nullable: true),
                Salary = table.Column<decimal>(type: "decimal(18,2)", nullable: false),
                                            Age = table.Column<int>(nullable: true)
                                                        },
                                                constraints: table =>
                                                        {
                                        table.PrimaryKey("PK_Employees", x => x.Id);
                                                        });
```

- **CreateTable**: يقوم بإنشاء جدول جديد في قاعدة البيانات اسمه **Employees**.
- **Columns**:
  - **Id**: يكون العمود الرئيسي (**Primary Key**) وجدول الهوية التلقائي.
  - **Name**: عمود نصي يبخزن اسم الموظف.
  - **Salary**: عمود رقمي يبخزن الراتب.
  - **Age**: عمود عددي يمكن أن يكون **null**.
- **PrimaryKey**: يحدد العمود **Id** كـ **Primary Key** للجدول.

Example of down method

- DropTable: بيقوم بحذف جدول Employees اللي تم إنشاؤه في الطريقة Up.

```
protected override void Down(MigrationBuilder migrationBuilder)
                                                                {
                                                migrationBuilder.DropTable(
                                                        name: "Employees");
                                                                }
```

لحد دلوقتى بردو ال database متعملتش فى sql server

تطبيق الـ Migration باستخدام Update-Database:

```
Update-Database
```

بعد تنفيذ الأمر Update-Database،ال Entity Framework هيقوم بتطبيق كل التغييرات اللي في الطريقة Up على قاعدة البيانات. وده هيشمل إنشاء الجداول، إضافة الأعمدة، وتحديد المفاتيح الأساسية.

الناتج النهائى فى قاعدة البيانات:

- هيتم إنشاء جدول Employees في قاعدة البيانات اللي اسمها CompanyDatabase (أو أي اسم قاعدة بيانات حددتة في الـ Connection String).
- الجدول هيكون فيه الأعمدة Id, Name, Salary, وAge بناءً على الكلاس Employee اللي أنشأته.
- "Add-Migration "InitialCreate بيقوم بإنشاء كود يمثل التغييرات اللي عايز تطبقها على قاعدة البيانات.
- Update-Database بيقوم بتطبيق التغييرات دي على قاعدة البيانات، وبيكون النتيجة هي تحديث قاعدة البيانات لتشمل الجداول الجديدة أو أي تعديلات أخرى.

يستمر العمل

ان شاء الله هيكون فى فايل تانى اكثر تفصيلا لكل الطرق