

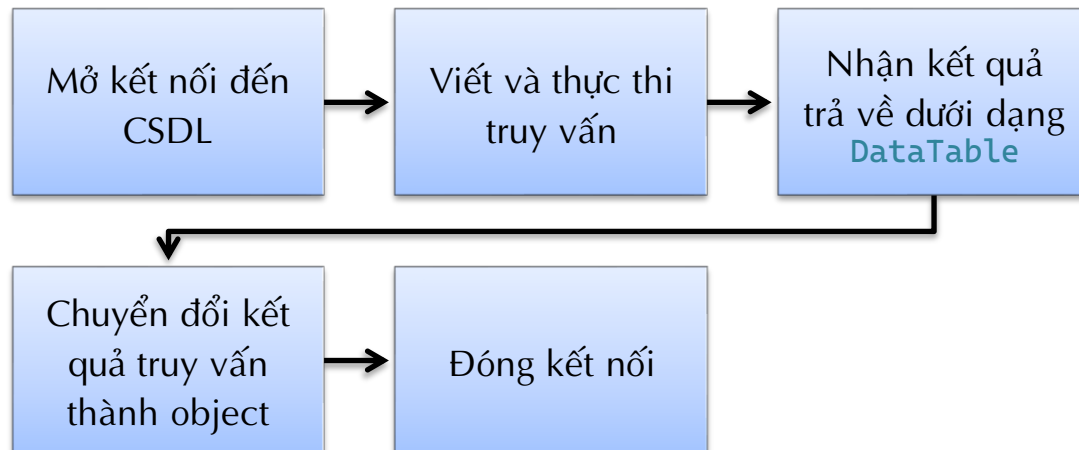
Buổi 3

Xây dựng cơ sở dữ liệu với Entity Framework

I. Entity Framework

1. Giới thiệu

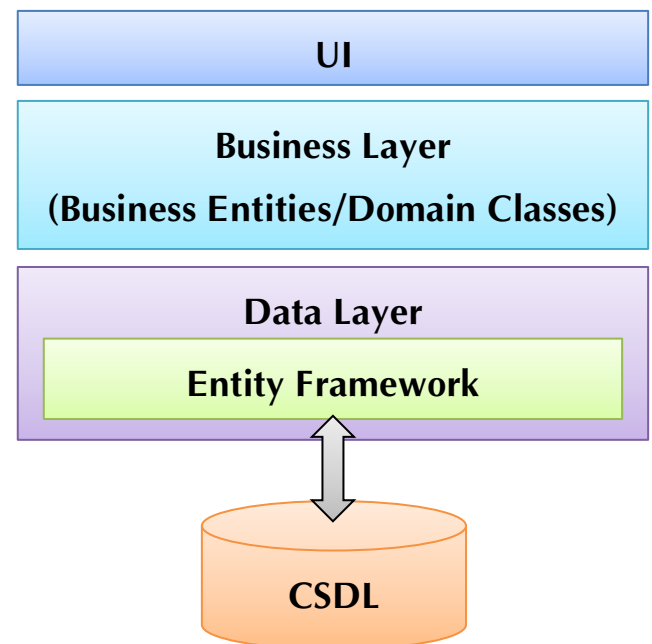
Trước phiên bản .NET Framework 3.5, để tương tác với cơ sở dữ liệu (CSDL), các ứng dụng thường sử dụng ADO.NET để lấy và cập nhật dữ liệu. Việc này phải trải qua các bước:



Quy trình ở trên khá rườm rà và dễ phát sinh lỗi. Bên cạnh đó, truy vấn thường được viết dưới dạng chuỗi và kết quả truy vấn ở dạng **DataTable** nên không tận dụng được các thế mạnh của lập trình hướng đối tượng.

Kể từ .NET Framework 3.5, Microsoft đã phát hành **Entity Framework** để tự động hóa các thao tác liên quan đến CSDL. Entity Framework (EF) là 1 framework ORM¹ giúp các lập trình viên .NET làm việc với CSDL đơn giản hơn bằng cách:

- Ánh xạ CSDL thành đối tượng trong C#.
- Giảm bớt việc phải viết các đoạn code mở kết nối và truy cập dữ liệu.
- Sử dụng kỹ thuật truy vấn bằng LINQ thân thiện với lập trình hướng đối tượng, thay vì phải viết truy vấn bằng SQL.



¹ Object Relational Mapping: Ánh xạ CSDL thành các đối tượng (object) trong ngôn ngữ lập trình.

2. Entity Framework Core

Kể từ phiên bản EF 6, **Entity Framework Core** được phát hành. Đây là 1 phiên bản mã nguồn mở, gọn nhẹ, đa nền tảng, thường được dùng trong các ứng dụng .NET Core, tuy nhiên nó vẫn có thể được dùng trong các ứng dụng .NET Framework 4.5 trở lên.

Tương tự như .NET Core, EF Core cũng có thể hoạt động đa nền tảng (Windows, Linux, MacOS) và kết nối đến nhiều hệ quản trị CSDL khác nhau (SQL Server, MySQL, SQLite...).

EF Core sử dụng 2 hướng tiếp cận là **Code-First** và **Database-First**:

- Code-First: Tạo các lớp đối tượng (model class) trước, sau đó EF Core sẽ tạo 1 CSDL có cấu trúc tương tự các lớp đối tượng này. Mọi thay đổi với model đều sẽ được ánh xạ sang CSDL bằng **Migration** (xem phần III):



- Database-First: Tạo CSDL trước, sau đó EF Core sẽ phát sinh các lớp đối tượng (model class) dựa trên cấu trúc CSDL này. Mọi thay đổi với CSDL đều sẽ được ánh xạ sang model bằng **Scaffolding**¹:

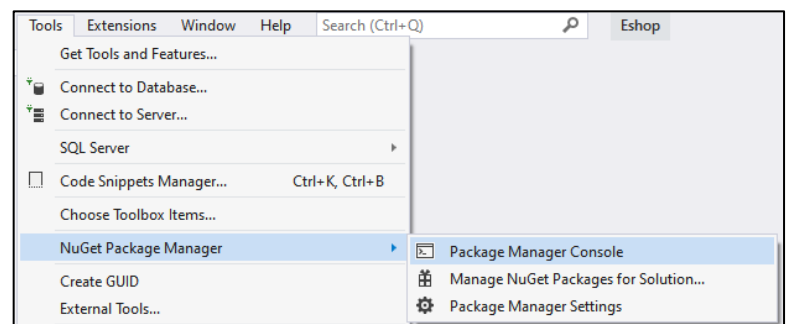


Môn học này sẽ sử dụng cách tiếp cận Code-First với hệ quản trị CSDL là SQL Server.

Để dùng được EF Core với SQL Server, cần cài đặt 1 **NuGet Package** đóng vai trò Database Provider đến SQL Server, có tên là **Microsoft.EntityFrameworkCore.SqlServer**.

Để cài đặt NuGet Package, có thể dùng 1 trong 2 cách sau:

- Cách 1: Cài đặt bằng console:
 - + **Tools → NuGet Package Manager → Package Manager Console (PMC):**



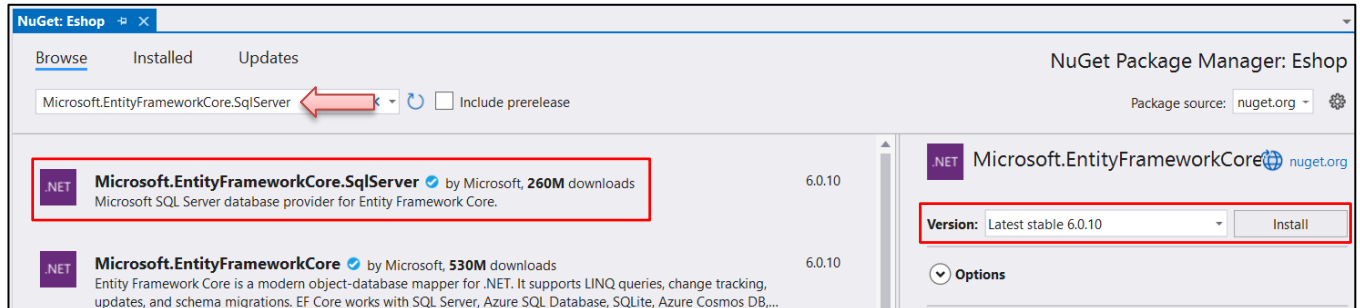
¹ Xem thêm về Database-First: <https://learn.microsoft.com/en-us/ef/core/managing-schemas/scaffolding>

- + Gõ câu lệnh sau vào PMC, sau đó nhấn phím **Enter**:

Install-Package Microsoft.EntityFrameworkCore.SqlServer

- Cách 2: Cài đặt bằng giao diện:

- + Click chuột phải vào project, chọn **Manage NuGet Packages...**
- + Mục Browse, tìm package **Microsoft.EntityFrameworkCore.SqlServer**
- + Cài đặt phiên bản ổn định mới nhất (Latest stable) của package này.



Lưu ý:

- .NET 6.0 tương thích với các NuGet Package có phiên bản là **6.x.x**.
- Việc cài đặt các NuGet Package cần phải có kết nối Internet.
- NuGet Package chỉ có tác dụng với project hiện tại.

3. Cơ chế hoạt động

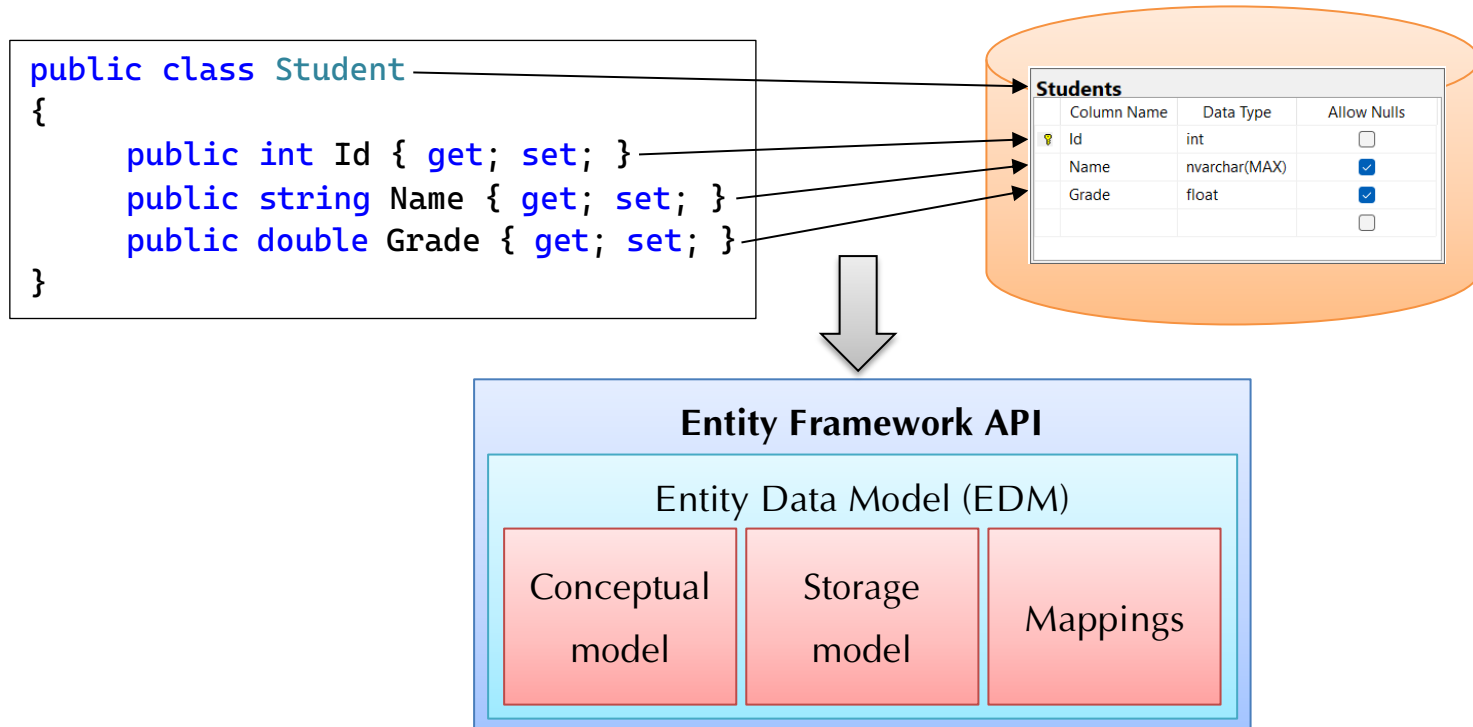
EF có API hỗ trợ các công việc sau:

- Ánh xạ (map) các thực thể (entity) của các lớp đối tượng trong model sang 1 CSDL (đối với Code-First) hoặc ngược lại (đối với Database-First).
- Dịch các câu truy vấn **LINQ**¹ sang SQL và thực thi chúng.
- Lưu vết (track) các thay đổi đối với thực thể và ánh xạ các thay đổi này sang CSDL bằng cách chuyển các thay đổi này thành các câu truy vấn INSERT, UPDATE, DELETE và thực thi chúng.

EF API xây dựng 1 **Entity Data Model** (EDM) thể hiện cấu trúc toàn bộ dữ liệu, gồm có:

- **Conceptual model**: Được xây dựng từ các entity class và đây là các đối tượng mà ta sẽ viết truy vấn trên đó.
- **Storage model**: Với hướng Code-First, Storage model sẽ dựa trên Conceptual model. Với hướng Database-First, Storage model sẽ dựa trên CSDL.
- **Mappings**: Ánh xạ giữa 2 loại model nói trên.

¹ LINQ sẽ được trình bày ở buổi 5



Với EF Core, việc truy cập dữ liệu sẽ sử dụng các model class, kèm theo đối tượng **Context** mô tả 1 phiên làm việc với CSDL. Đối tượng này đóng vai trò là cầu nối trung gian giữa ứng dụng ASP.NET Core và CSDL, cho phép truy vấn, cập nhật dữ liệu và có các đặc điểm sau:

- Kế thừa từ class **DbContext** (thuộc namespace **Microsoft.EntityFrameworkCore**).
- Ánh xạ bảng trong CSDL thành collection kiểu **DbSet<T>**, trong đó T là model class.
- Lưu trữ dữ liệu của mỗi dòng trong bảng thành 1 object trong collection nói trên.
- Mỗi cột trong bảng sẽ thể hiện bằng thuộc tính của model class T.

Khi đã xây dựng được liên kết ánh xạ giữa CSDL và model, lập trình viên có thể truy vấn dữ liệu bằng LINQ thay vì SQL, do đó có thể dễ dàng thực hiện các thao tác CRUD¹ bằng code C# mà không cần viết 1 câu truy vấn SQL nào.

II. Thiết lập project dùng EF Code-First

Đối với hướng tiếp cận Code-First, lập trình viên sẽ bắt đầu bằng việc code² để tạo ra các model class thay vì xây dựng cơ sở dữ liệu trên SQL Server hoặc 1 hệ quản trị CSDL nào đó (tất nhiên vẫn phải dựa trên bản phân tích CSDL đã có). Sau đó, dựa vào các model class này, EF API sẽ tạo ra CSDL có cấu trúc giống như vậy.

Trong code mẫu minh họa ở buổi 3, chúng ta sẽ xây dựng model class để tạo ra CSDL có tên là **Eshop**. Do đó, ta sẽ đặt tên project là **Eshop**.

¹ Thao tác CRUD sẽ được trình bày ở buổi 4

² Đây là lý do vì sao cách tiếp cận này có tên là Code-First

Các bước thiết lập cho project để có thể dùng được EF Code-First như sau:

- Bước 1 – Tạo chuỗi kết nối đến CSDL: Mở file **appsettings.json** và bổ sung chuỗi kết nối có tên là **Eshop** như sau, trong đó mục **Server** và **Database** lần lượt là tên máy chủ SQL và tên CSDL:

```
"ConnectionStrings": {  
    "Eshop": "Server=localhost;Database=Eshop;Trusted_Connection=  
True;MultipleActiveResultSets=true"  
}
```

- Bước 2 – Xây dựng **Context**: Tạo thư mục **Data** trong project, trong đó tạo class **EshopContext** kế thừa từ class **DbContext**, kèm theo 1 phương thức khởi tạo cũng kế thừa từ phương thức khởi tạo tương ứng của **DbContext**:

```
using Microsoft.EntityFrameworkCore;  
  
namespace Eshop.Data  
{  
    public class EshopContext : DbContext  
    {  
        public EshopContext(DbContextOptions<EshopContext>  
options): base(options)  
        { }  
    }  
}
```

- Bước 3 – Thiết lập service: Bổ sung câu lệnh sau vào phần thiết lập dịch vụ của file **Program.cs** để quy định project sẽ sử dụng **EshopContext** như 1 service với chuỗi kết nối có tên là **Eshop** từ file **appsettings.json**:

```
builder.Services.AddDbContext<EshopContext>  
    (options => options.UseSqlServer  
        (builder.Configuration.GetConnectionString("Eshop")));
```

Sau này, việc tạo CSDL lần đầu tiên từ model cũng như cập nhật CSDL khi model thay đổi sẽ được giải quyết bằng **Migration**.

III. Migration

1. Khái niệm

Trước đây, EF sử dụng các bộ khởi tạo (database initializer) như sau để đảm bảo tạo ra được CSDL tương ứng với model:

- **CreateDatabaseIfNotExist**: Tạo mới CSDL nếu chưa tồn tại.
- **DropCreateDatabaseIfModelChanges**: Xóa và tạo lại CSDL nếu model thay đổi.
- **DropCreateDatabaseAlways**: Luôn xóa và tạo lại CSDL khi chạy ứng dụng.

Tuy nhiên, nhược điểm của các bộ khởi tạo nói trên là chúng luôn xóa CSDL cũ và tạo lại CSDL mới, khiến cho dữ liệu, và các đối tượng khác trong CSDL như stored procedure, function, trigger... đều bị mất đi.

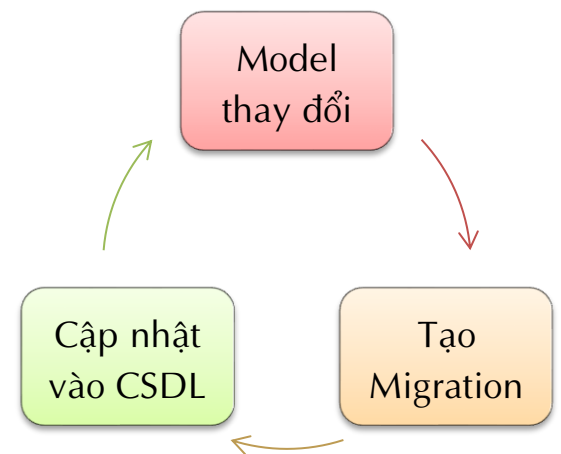
Migration là 1 chức năng trong EF Code-First giúp cập nhật cấu trúc CSDL nếu model thay đổi mà không làm mất dữ liệu và các đối tượng khác trong đó.

Migration cung cấp một bộ các công cụ cho phép:

- Khởi tạo CSDL lần đầu tương ứng với model.
- Tạo ra những lần migration để lưu vết (track) mỗi khi model thay đổi.
- Cập nhật cấu trúc CSDL theo những thay đổi này.

Quy trình sử dụng migration như sau:

- Mỗi khi model có sự thay đổi (kể cả lần khởi tạo đầu tiên), cần tạo 1 lần migration, hiểu đơn giản đó là 1 *phiên bản*.
- Cập nhật phiên bản migration này vào CSDL. EF sẽ tự nhận biết CSDL đang ở phiên bản nào và tự động cập nhật lên phiên bản mới nhất.



2. Khởi tạo CSDL lần đầu

Để dùng được migration, cần cài package sau: **Microsoft.EntityFrameworkCore.Tools**.

Trong ví dụ ở phần III.2 này, giả sử cần tạo ra bảng

Accounts với cấu trúc như hình bên:

Accounts	
Id	int
Username	nvarchar(MAX)
Password	nvarchar(MAX)

Đầu tiên, tạo class **Account** trong thư mục **Models**:

```
public class Account
{
    public int Id { get; set; }
    public string Username { get; set; }
    public string Password { get; set; }
}
```

Tiếp theo, trong class **EshopContext**, cần khai báo model **Account** dưới dạng 1 thuộc tính của class này, với kiểu dữ liệu là **DbSet<T>** bằng cách bổ sung câu lệnh sau:

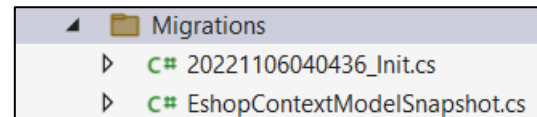
```
public class EshopContext : DbContext
{
    // .....
    public DbSet<Account> Accounts { get; set; }
}
```

Để tạo migration, gõ lệnh sau vào PMC:

Add-Migration Init

Trong đó: **Init** là tên của lần migration hiện tại do lập trình viên tự đặt.

Sau khi thực hiện câu lệnh này, thư mục **Migrations** được tạo ra, chứa 2 file sau:



- **<timestamp>_Init.cs**, với <timestamp> là ngày giờ tạo ra file này. Trong file có chứa class **Init**, gồm phương thức **Up()** chứa code để cập nhật CSDL lên phiên bản này, và phương thức **Down()** dùng để hoàn trả (revert) những thay đổi của **Up()**. Code trong file migration được viết bằng Fluent API.
- **EshopContextModelSnapshot.cs**: Chứa class cùng tên, là tổng thể cấu trúc của model tính đến lần migration hiện tại (được gọi là **model snapshot**).

Sau khi migration đã được tạo ra, gõ lệnh sau vào PMC để tạo ra CSDL lần đầu:

Update-Database

CSDL **Eshop** được tạo ra trong server **localhost** với bảng **Accounts** có cấu trúc tương tự như model **Account** đã tạo ở trên:

Accounts			
	Column Name	Data Type	Allow Nulls
🔑	Id	int	<input type="checkbox"/>
	Username	nvarchar(MAX)	<input checked="" type="checkbox"/>
	Password	nvarchar(MAX)	<input checked="" type="checkbox"/>
			<input type="checkbox"/>

Ngoài ra, trong CSDL được tạo ra bằng Migration luôn có 1 bảng **__EFMigrationsHistory** lưu lại lịch sử những lần Migration. Dựa vào bảng này, EF có thể biết được hiện tại CSDL còn thiếu những phiên bản Migration nào. Dữ liệu của bảng này hiện tại như sau:

	MigrationId	ProductVersion
▶	20221106040436_Init	6.0.10
*	NULL	NULL

3. Cập nhật Migration khi model thay đổi

Qua thời gian ứng dụng phát triển, model có thể có sự thay đổi, và lập trình viên phải đảm bảo CSDL luôn được cập nhật đúng với model.

Do đó, mỗi khi model thay đổi, lập trình viên phải thực hiện 2 bước sau:

- Bước 1: Chạy lệnh **Add-Migration** trong PMC. Do EF đã tự động thêm thời gian tạo migration nên tên migration thường chỉ cần mô tả ngắn gọn thay đổi trong lần cập nhật này, ví dụ:

Add-Migration AddModelProduct

Add-Migration EditModelProduct_AddExpiringDate

.....

- Bước 2: EF sẽ so sánh cấu trúc model với model snapshot để đối chiếu thay đổi, sau đó tạo ra 1 migration phiên bản mới chỉ chứa những điểm khác biệt của model

so với model snapshot hiện tại. Cần kiểm tra kỹ phương thức `Up()` của migration vừa tạo ra để đảm bảo là phiên bản này chứa đúng những gì cần cập nhật.

- Bước 3: Chạy lệnh **Update-Database** trong PMC để cập nhật thay đổi từ migration vào CSDL. EF sẽ dựa vào bảng **__EFMigrationsHistory** để xác định phiên bản migration hiện tại của CSDL, sau đó cập nhật tất cả những phiên bản migration còn thiếu.

Ví dụ: Bổ sung thuộc tính vào 1 model class đã có:

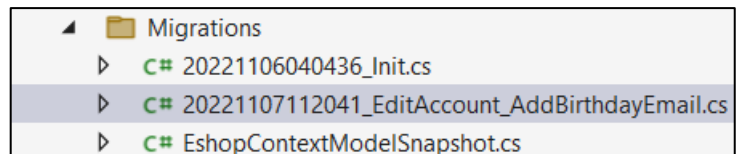
- Thêm thuộc tính **Birthday** và **Email** vào class **Account**:

```
public class Account
{
    .....
    public DateTime Birthday { get; set; }
    public string Email { get; set; }
}
```

- Lần lượt chạy 2 lệnh sau trong PMC:

Add-Migration EditAccount_AddBirthdayEmail
Update-Database

- Sau khi chạy lệnh **Add-Migration**, EF sẽ tạo ra 1 file migration mới, trong đó phương thức `Up()` chứa code bổ sung cột **Birthday** và **Email** vào bảng **Accounts**:



```
protected override void Up(MigrationBuilder migrationBuilder)
{
    migrationBuilder.AddColumn<DateTime>(
        name: "Birthday",
        table: "Accounts",
        type: "datetime2",
        nullable: false,
        defaultValue: new DateTime(1, 1, 1, 0, 0, 0, 0, DateTimeKind.Unspecified));

    migrationBuilder.AddColumn<string>(
        name: "Email",
        table: "Accounts",
        type: "nvarchar(max)",
        nullable: true);
}
```

- Sau khi chạy lệnh **Update-Database**, CSDL sẽ được cập nhật lại theo model:

Accounts			
	Column Name	Data Type	Allow Nulls
🔑	Id	int	<input type="checkbox"/>
	Username	nvarchar(MAX)	<input checked="" type="checkbox"/>
	Password	nvarchar(MAX)	<input checked="" type="checkbox"/>
	Birthday	datetime2(7)	<input type="checkbox"/>
	Email	nvarchar(MAX)	<input checked="" type="checkbox"/>
			<input type="checkbox"/>

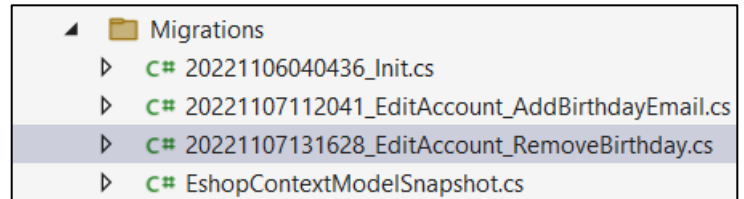
Ví dụ: Xóa thuộc tính khỏi 1 model class đã có:

- Xóa thuộc tính **Birthday** khỏi class **Account**.
- Lần lượt chạy 2 lệnh sau trong PMC:

Add-Migration EditAccount_RemoveBirthday
Update-Database

- Sau khi chạy lệnh **Add-Migration**, EF sẽ tạo ra 1 file migration mới, trong đó phương thức **Up()** chứa code loại bỏ cột **Birthday** khỏi bảng **Accounts**:

```
protected override void Up(MigrationBuilder migrationBuilder)
{
    migrationBuilder.DropColumn(
        name: "Birthday",
        table: "Accounts");
}
```



- Sau khi chạy lệnh **Update-Database**, CSDL sẽ được cập nhật lại theo model:

Accounts			
	Column Name	Data Type	Allow Nulls
🔑	Id	int	<input type="checkbox"/>
	Username	nvarchar(MAX)	<input checked="" type="checkbox"/>
	Password	nvarchar(MAX)	<input checked="" type="checkbox"/>
	Email	nvarchar(MAX)	<input checked="" type="checkbox"/>
			<input type="checkbox"/>

Ví dụ: Thêm class vào model:

- Thêm class **Product** vào thư mục **Models** như sau:

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

- Khai báo model **Product** vào **EshopContext**:

```
public class EshopContext : DbContext
{
    // .....
    public DbSet<Account> Accounts { get; set; }
    public DbSet<Product> Products { get; set; }
}
```

- Lần lượt chạy 2 lệnh sau trong PMC tương tự như 2 ví dụ ở trên:

Add-Migration AddProduct

Update-Database

Lưu ý: EF sẽ chỉ làm việc với những model class đã được khai báo trong **Context**.

4. Loại bỏ Migration

EF cung cấp lệnh **Remove-Migration** để loại bỏ phiên bản migration mới nhất. Khi chạy lệnh này, EF sẽ thực hiện phương thức **Down()** của phiên bản migration mới nhất để hoàn trả model snapshot về tình trạng như trước khi áp dụng phiên bản migration này, sau đó sẽ xóa phiên bản migration này.

Lưu ý:

- Lệnh **Remove-Migration** chỉ loại bỏ phiên bản migration mới nhất.
- Chỉ có thể loại bỏ phiên bản migration chưa được áp dụng vào CSDL.
- Không được xóa file migration thủ công mà phải thực hiện loại bỏ bằng lệnh trên.

IV. Các quy chuẩn của EF Code-First

Trong các ví dụ trên, EF đã tạo ra các bảng và tự thiết lập khóa chính, kiểu dữ liệu của các cột trong bảng... dựa trên model class mà không cần lập trình viên phải thiết lập thủ công. Đó là do EF Code-First đã có những quy chuẩn (**convention**) cho việc này.

1. Khóa chính (Primary key) và khóa ngoại (Foreign key)

Các model class trong EF chỉ có 1 khóa dùng để phân biệt các đối tượng khác nhau của class đó. Quy chuẩn EF quy định thuộc tính khóa có tên là **Id** hoặc là **<tên class>Id** (không phân biệt hoa/thường). Khi ánh xạ sang bảng trong CSDL, thuộc tính này sẽ trở thành khóa chính của bảng.

Nếu muốn thuộc tính khóa có tên khác với quy tắc trên, cần sử dụng **Data Annotation**¹.

Nếu thuộc tính khóa của model class có kiểu dữ liệu là **int**, khóa chính tương ứng sẽ tự động có tính chất **Identity(1, 1)**².

Ví dụ:

- Class **Product** có khóa là **Id** → Bảng **Products** có khóa chính là **Id**:

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

¹ Xem phần VI

² Số nguyên tự động tăng dần từ 1, bước tăng là 1

- Class **Student** có khóa là **StudentId** → Bảng **Students** có khóa chính là **StudentId**:

```
public class Student
{
    public int StudentId { get; set; }
}
```

Quy chuẩn về khóa ngoại sẽ được trình bày ở phần V.

2. Bảng (Table) và cột (Column)

EF sẽ ánh xạ model class thành CSDL theo các quy chuẩn sau:

Quy chuẩn	Giải thích
Schema	CSDL được tạo với schema mặc định là dbo .
Tên bảng	Là tên của DbSet tương ứng trong Context, và mặc định là danh từ số nhiều của tên class. Do đó, nên đặt tên class và tên thuộc tính là danh từ bằng tiếng Anh. <u>Ví dụ:</u> <ul style="list-style-type: none"> – Class Account → Bảng Accounts – Class Box → Bảng Boxes
Tên cột	Là tên thuộc tính của class
Thứ tự cột	Là thứ tự thuộc tính của class. Tuy nhiên, cột khóa chính sẽ được đưa lên đầu tiên.
Cột cho phép Null (Allow Nulls)	Nếu thuộc tính của class thuộc kiểu Reference Type ¹ (tham chiếu) hoặc kiểu Nullable (dữ liệu có thể mang giá trị null), cột tương ứng trong bảng được đánh dấu Allow Nulls .
Cột không cho phép Null	Nếu thuộc tính của class thuộc kiểu Value Type , cột tương ứng trong bảng sẽ là Not Null .

Các kiểu dữ liệu của .NET được chia làm 2 loại chính: **Value Type** và **Reference Type**.

Các kiểu Reference Type (ví dụ: **string**, class, interface, delegate...) có thể nhận giá trị **null**. Do đó, EF tự động đánh dấu **Allow Nulls** cho cột tương ứng của bảng.

Các kiểu Value Type (ví dụ: **int**, **bool**, **double**, **DateTime**, enum, struct...) không thể nhận giá trị **null**. Do đó, EF **không** đánh dấu **Allow Nulls** cho cột tương ứng của bảng.

Khi tạo model class, có thể đặt dấu ? sau kiểu dữ liệu để cho phép 1 thuộc tính nhận giá trị **null** (ngay cả khi kiểu dữ liệu là Value Type). EF cũng tự động đánh dấu cột tương ứng trong bảng là **Allow Nulls**.

¹ Xem thêm về các kiểu dữ liệu của C# tại đây: <https://www.geeksforgeeks.org/c-sharp-data-types>

Ví dụ:

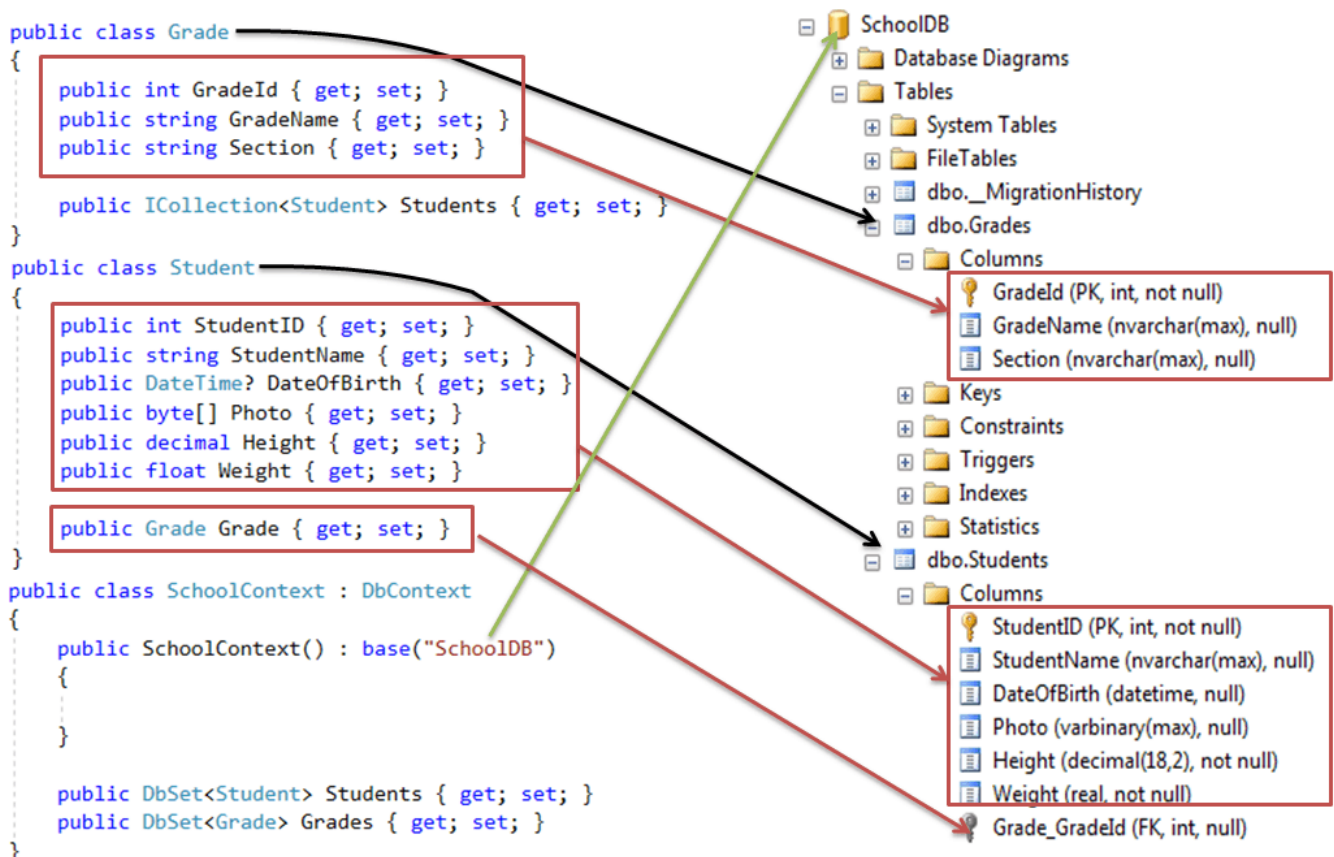
```
public class Account
{
    public int? Age { get; set; } // Cả 2 thuộc tính có thể null
    public bool? Status { get; set; } // mặc dù là Value Type
}
```

3. Kiểu dữ liệu

Không phải tất cả kiểu dữ liệu của .NET đều có kiểu dữ liệu tương ứng ở SQL Server và ngược lại. Do đó, quy chuẩn ánh xạ các kiểu dữ liệu thông dụng như sau:

Kiểu dữ liệu .NET	Kiểu dữ liệu SQL Server
Int32, int	int
Int64, long	bigint
float	real
double	float
Boolean, bool	bit
DateTime	datetime2
String, string	nvarchar(MAX)
char	Không có
object	

Ví dụ: Minh họa về việc ánh xạ model class thành CSDL theo các quy chuẩn trên:



V. Tạo khóa ngoại

Trong quá trình xây dựng CSDL, khóa ngoại (foreign key) được tạo ra do quan hệ 1-N giữa 2 thực thể (quan hệ N-N cũng được tách thành 2 quan hệ 1-N). Khi sử dụng EF Code-First để tạo ra CSDL, có 2 cách để tạo khóa ngoại giữa 2 bảng, đó là xây dựng model theo quy chuẩn (convention) hoặc sử dụng code Fluent API¹.

Giả sử cần tạo khóa ngoại cho 2 bảng **Products** và **ProductTypes** như hình bên. Mỗi quan hệ 1-N này được phát biểu như sau: "1 sản phẩm thuộc 1 loại sản phẩm, nhưng 1 loại sản phẩm có thể có nhiều sản phẩm". Có thể hiểu rằng: bảng **Products** sẽ tham chiếu (reference) tới bảng **ProductTypes**.

Tài liệu này sẽ chỉ trình bày về việc tạo khóa ngoại bằng cách xây dựng model theo 1 trong 4 quy chuẩn sau:

1. Quy chuẩn 1

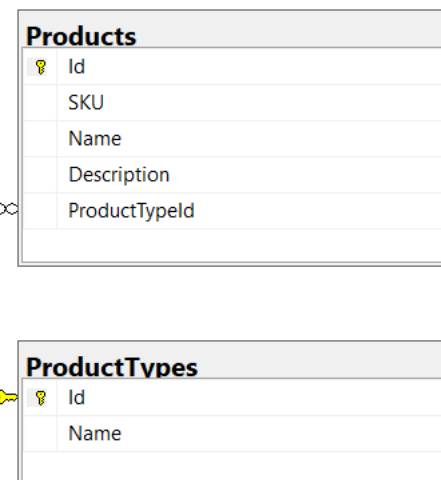
Bổ sung 1 thuộc tính có kiểu **ProductType** và có tên là **ProductType** vào class **Product**. Thuộc tính này được gọi là *Reference navigation property*, cho EF biết rằng class **Product** đang tham chiếu đến 1 đối tượng của class **ProductType**:

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

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

Reference
navigation property

Khi tạo migration và ánh xạ vào CSDL, EF sẽ tự bổ sung cột **ProductTypeId** trong bảng **Products** làm khóa ngoại tham chiếu đến bảng **ProductTypes**.



¹ Xem thêm về cách tạo khóa ngoại bằng Fluent API: <https://www.entityframeworktutorial.net/code-first/configure-one-to-many-relationship-in-code-first.aspx#configure-one-to-many-using-fluent-api>

2. Quy chuẩn 2

Bổ sung 1 thuộc tính có kiểu `List<Product>` và có tên là **Products** vào class `ProductType`. Thuộc tính này được gọi là *Collection navigation property*, cho EF biết rằng class `ProductType` đang chứa nhiều đối tượng của class `Product`:

```
public class ProductType
{
    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 SKU { get; set; }
    public string Name { get; set; }
    public string Description { get; set; }
}
```

Collection
navigation property

Khi tạo migration và ánh xạ vào CSDL, EF sẽ tự bổ sung cột **ProductTypeId** trong bảng **Products** làm khóa ngoại tham chiếu đến bảng **ProductTypes**.

3. Quy chuẩn 3

Kết hợp cả quy chuẩn 1 và 2 bằng cách bổ sung cả Reference navigation property vào class `Product` và Collection navigation property vào class `ProductType`:

```
public class ProductType
{
    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 SKU { get; set; }
    public string Name { get; set; }
    public string Description { get; set; }
    public ProductType ProductType { get; set; }
}
```

4. Quy chuẩn 4

Nhược điểm của cả 3 cách ở trên là class **Product** không chứa thuộc tính khóa ngoại **ProductTypeId** (mặc dù trong bảng **Products** có cột này), do đó có thể gây khó khăn khi muốn truy cập thuộc tính **ProductTypeId** sau này.

Quy chuẩn 4 tương tự quy chuẩn 3 nhưng bổ sung thêm thuộc tính khóa ngoại **ProductTypeId** vào class **Product**. Thuộc tính này sẽ có cùng kiểu dữ liệu với thuộc tính khóa chính của class **ProductType** (hiện tại là **int**):

```
public class ProductType
{
    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 SKU { get; set; }
    public string Name { get; set; }
    public string Description { get; set; }
    public int ProductTypeId { get; set; }
    public ProductType ProductType { get; set; }
}
```

Cách 4 định nghĩa khóa ngoại bằng 2 navigation property ở cả 2 đầu, kèm theo cả thuộc tính khóa ngoại, do đó đây là cách đầy đủ nhất, thuận tiện nhất khi viết code truy vấn dữ liệu sau này. Khi xây dựng khóa ngoại, **khuyến cáo** sử dụng cách 4 này.

5. Quy chuẩn đặt tên khi xây dựng khóa ngoại

Giả sử có mối quan hệ class **B** tham chiếu đến class **A** (tức là $B \rightarrow A$):

- Reference navigation property đặt ở class **B**, có kiểu dữ liệu là **A** và có tên là **A**.
- Collection navigation property đặt ở class **A**, có kiểu dữ liệu là **List** và có tên là danh từ số nhiều của **B**.
- Thuộc tính khóa ngoại đặt ở class **B**, có kiểu dữ liệu trùng với kiểu dữ liệu khóa chính của class **A** và có tên là **AId** (A kèm theo "Id").

VI. Data Annotation

Trong .NET Framework và sau này là .NET Core, **Data Annotation** dùng để thêm các chú thích mở rộng vào model thông qua các thẻ thuộc tính. Tính năng này được Microsoft

giới thiệu lần đầu ở .NET Framework 3.5. Kết hợp với EF Code-First, Data Annotation cho phép định nghĩa model chi tiết hơn, dễ dàng hơn mà không cần dùng đến Fluent API; cho phép cài đặt model khác với quy chuẩn của EF; giúp kiểm tra dữ liệu nhập vào từ người dùng cũng như thay đổi cách dữ liệu hiển thị trên giao diện.

Để sử dụng được Data Annotation, cần sử dụng các namespace sau:

```
System.ComponentModel
System.ComponentModel.DataAnnotations
System.ComponentModel.DataAnnotations.Schema
```

Các thuộc tính Data Annotation được phân chia thành 3 loại chính:

- Thuộc tính mô hình (**Modelling Attribute**): định nghĩa model và các thuộc tính của model, cho phép cài đặt model khác với quy chuẩn của EF.
- Thuộc tính hiển thị (**Display Attribute**): thay đổi cách hiển thị dữ liệu trên view.
- Thuộc tính kiểm tra (**Validation Attribute**): dùng để thêm các quy luật để kiểm tra (validate) dữ liệu.

Các thuộc tính này được đặt trong cặp dấu ngoặc [] và đặt phía trước nơi cần chú thích.

Lưu ý: Để tránh nhầm lẫn về thuật ngữ, trong tài liệu này, các thuộc tính Data Annotation (Data Annotation attribute) sẽ được gọi là chú thích (annotation).

1. Thuộc tính mô hình

a) Chú thích [Key]

Mặc định, EF sẽ lấy thuộc tính Id hoặc <tên class>Id làm khóa chính. Chú thích [Key] cho tên thuộc tính giúp quy định khóa chính tùy ý mà không cần tuân theo quy chuẩn này.

Ví dụ: Với class Account, quy định khóa chính là Username:

```
public class Account
{
    [Key]
    public string Username { get; set; }
}
```

b) Chú thích [Index]

Chú thích [Index] dùng để tạo ràng buộc index trên 1 thuộc tính.

Cú pháp: [Index(nameof(property_name), IsUnique = bool?, Name = string?)] cho model class. Trong đó:

- Tham số **property_name** là tên thuộc tính muốn chỉ định làm index (index có thể bao gồm nhiều thuộc tính).

- Tham số **IsUnique** cho biết ràng buộc index có duy nhất hay không (mặc định ràng buộc index trong EF không mang tính duy nhất).
- Tham số **Name** cho phép tự đặt tên ràng buộc index.

Ví dụ:

- Chỉ định thuộc tính **Email** là unique index của model **Account**:

```
[Index(nameof(Email), IsUnique = true)]
public class Account { ... }
```

- Chỉ định thuộc tính **Username** và **Phone** là index của model **Account** và đặt tên ràng buộc index là **index_Account**:

```
[Index(nameof(Username), nameof(Phone), Name = "index_Account")]
public class Account { ... }
```

c) Chú thích [Foreign Key]

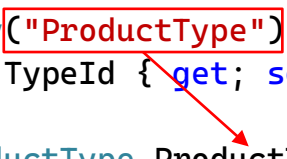
Mặc định, EF sẽ lấy thuộc tính có tên là <tên_class_được_tham_chiếu>Id làm khóa ngoại. Trong ví dụ trên, thuộc tính **ProductTypeId** của class **Product** là khóa ngoại. Chú thích [ForeignKey] giúp quy định khóa ngoại tùy ý mà không cần tuân theo quy chuẩn này.

Giả sử ta muốn đặt tên thuộc tính khóa ngoại là **TypeId**. Để EF nhận ra đây là khóa ngoại, chúng ta có thể chú thích theo 1 trong 3 cách sau:

- Cách 1: Chú thích ở thuộc tính khóa ngoại cho biết thuộc tính nào là reference navigation property:

```
public class Product
{
    .....
    [ForeignKey("ProductType")]
    public int TypeId { get; set; }

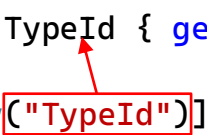
    public ProductType ProductType { get; set; }
}
```



- Cách 2: Chú thích ở reference navigation property cho biết thuộc tính nào là thuộc tính khóa ngoại:

```
public class Product
{
    .....
    public int TypeId { get; set; }


    [ForeignKey("TypeId")]
    public ProductType ProductType { get; set; }
}
```



- Cách 3: Chú thích ở collection navigation property cho biết thuộc tính nào là thuộc tính khóa ngoại:

```
public class Product
{
    .....
    public int TypeId { get; set; }
    public ProductType ProductType { get; set; }
}

public class Account
{
    .....
    [ForeignKey("TypeId")]
    public List<Product> Products { get; set; }
}
```



d) Chú thích [Table]

Theo quy chuẩn, class được khai báo trong **Context** sẽ được tạo thành bảng trong CSDL, và tên bảng là danh từ số nhiều của tên class (ví dụ: class **Account** → bảng **Accounts**). Chú thích [Table] dành cho class giúp quy định tên bảng tùy ý mà không cần tuân theo quy chuẩn này.

Cú pháp: [Table(string name, Schema = string?)], trong đó tham số name là tên bảng và tham số Schema là tên schema (schema mặc định là **dbo**).

Ví dụ:

- Ánh xạ class **Product** thành bảng **SanPham**:

```
[Table("SanPham")]
public class Product { ... }
```

- Ánh xạ class **Account** thành bảng **TaiKhoan**, schema là **admin**:

```
[Table("TaiKhoan", Schema = "admin")]
public class Account { ... }
```

e) Chú thích [Column]

Theo quy chuẩn, EF sẽ lấy tên thuộc tính của class để làm tên cột, thứ tự thuộc tính sẽ là thứ tự cột, kiểu dữ liệu của thuộc tính sẽ quyết định kiểu dữ liệu của cột. Chú thích [Column] dành cho thuộc tính giúp tự quy định tên, thứ tự, kiểu dữ liệu cho cột mà không cần tuân theo quy chuẩn này.

Cú pháp: [Column (string name, Order = int?, TypeName = string?)], trong đó tham số name là tên cột, tham số Order và TypeName là thứ tự cột (bắt đầu từ 0) và kiểu dữ liệu của cột đó.

Ví dụ:

- Ánh xạ thuộc tính Phone thành cột **SDT**:

```
[Column("SDT")]  
public string Phone { get; set; }
```

- Ánh xạ thuộc tính Birthday thành cột **NgàySinh**, là cột thứ 3 trong bảng và kiểu dữ liệu là DateTime2:

```
[Column("NgàySinh", Order = 3, TypeName = "DateTime2")]  
public DateTime Birthday { get; set; }
```

f) Chú thích [NotMapped]

Mặc định, EF sẽ ánh xạ tất cả thuộc tính của class thành cột trong bảng. Chú thích [NotMapped] cho phép quy định 1 thuộc tính nào đó sẽ không được ánh xạ sang CSDL.

Ví dụ: Chỉ định thuộc tính Note không được ánh xạ sang CSDL:

```
[NotMapped]  
public string Note { get; set; }
```

Lưu ý: Các thuộc tính mô hình đều có ảnh hưởng đến CSDL, do đó khi sử dụng các thuộc tính loại này, cần phải thực hiện lại thao tác migration.

2. Thuộc tính hiển thị

Mặc định, EF sẽ lấy tên thuộc tính của class làm tên hiển thị trên view (yêu cầu view phải sử dụng ViewModel). Chú thích [DisplayName] dùng để quy định tên hiển thị của thuộc tính.

Cú pháp: [DisplayName(string name)], trong đó tham số name là tên hiển thị.

Ví dụ: Thay đổi tên hiển thị cho thuộc tính Username thành "Tên đăng nhập":

- Chú thích cho thuộc tính Username của class Account như sau:

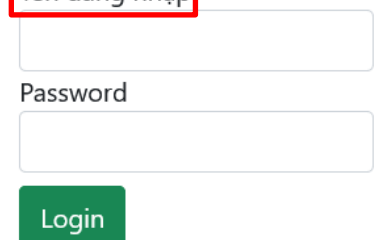
```
[DisplayName("Tên đăng nhập")]  
public string Username { get; set; }
```

- Tạo view Login trong đó chứa form đăng nhập có câu lệnh hiển thị tên đăng nhập:

```
<label asp-for="Username"></label>
```



Lưu ý: Việc thay đổi tên hiển thị chỉ có ảnh hưởng đối với view, không ảnh hưởng đến CSDL, do đó không cần phải thực hiện lại thao tác migration.



3. Thuộc tính kiểm tra

Các chú thích thuộc nhóm này cho phép đặt ra các điều kiện cho dữ liệu và kiểm tra (validate) để đảm bảo là người dùng nhập dữ liệu đúng yêu cầu. Mỗi chú thích đều có tham số ErrorMessage để cài đặt thông báo lỗi khi người dùng nhập sai điều kiện.

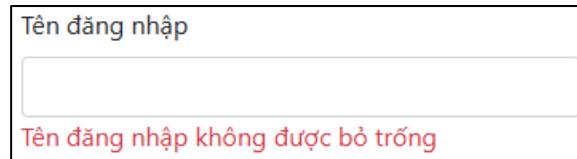
a) Chú thích [Required]

Chú thích [Required] quy định thuộc tính này bắt buộc phải nhập, không được bỏ trống.

Cú pháp: [Required(ErrorMessage = string?)]

Ví dụ: Bắt buộc người dùng phải nhập Username, nếu bỏ trống sẽ hiển thị thông báo lỗi:

```
[Required(ErrorMessage = "Tên đăng nhập không được bỏ trống")]  
public string Username { get; set; }
```



The image shows a web form with a text input field labeled "Tên đăng nhập". Below the input field, there is a red error message: "Tên đăng nhập không được bỏ trống".

b) Chú thích [MinLength] và [MaxLength]

Chú thích [MinLength] và [MaxLength] cho phép quy định độ dài tối thiểu và tối đa của thuộc tính có dạng chuỗi.

Cú pháp: [MinLength(int length, ErrorMessage = string?)], trong đó tham số length là độ dài tối thiểu. Chú thích [MaxLength] có cú pháp tương tự.

Mặc định EF sẽ ánh xạ các thuộc tính kiểu string sang các cột có kiểu nvarchar(MAX). Tuy nhiên nếu thuộc tính có chú thích [MaxLength(X)] sẽ tạo ra cột có kiểu nvarchar(X).

Ví dụ: Quy định thuộc tính Username có độ dài từ 6 đến 20 ký tự:

```
[MinLength(6, ErrorMessage = "Tên đăng nhập tối thiểu 6 ký tự")]  
[MaxLength(20, ErrorMessage = "Tên đăng nhập tối đa 20 ký tự")]  
public string Username { get; set; }
```

c) Chú thích [StringLength]

Chú thích [StringLength] cho phép quy định độ dài tối thiểu và tối đa của thuộc tính có dạng chuỗi.

Cú pháp: [StringLength(int length, MinimumLength = int?, ErrorMessage = string?)], trong đó tham số length là độ dài tối đa và tham số MinimumLength là độ dài tối thiểu. Có thể xem chú thích này là tổng hợp của [MinLength] và [MaxLength].

Ví dụ: Quy định thuộc tính Username có độ dài từ 6 đến 20 ký tự:

```
[StringLength(20, MinimumLength = 6, ErrorMessage = "Tên đăng nhập từ  
6-20 ký tự")]  
public string Username { get; set; }
```

d) Chú thích [EmailAddress]

Chú thích [EmailAddress] quy định thuộc tính có định dạng email. Khi dùng Tag Helper để tạo thẻ <input>, thuộc tính này sẽ tạo ra 1 thẻ <input type="email">.

Cú pháp: [EmailAddress(ErrorMessage = string?)].

e) Chú thích [Range]

Chú thích [Range] quy định khoảng giá trị cho 1 thuộc tính.

Cú pháp dành cho kiểu int, kiểu double và kiểu bất kỳ:

```
[Range(int min, int max, ErrorMessage = string?)]  
[Range(double min, double max, ErrorMessage = string?)]  
[Range(Type type, string min, string max, ErrorMessage = string?)]
```

Ví dụ:

- Thuộc tính Age nằm trong khoảng 18-55:

```
[Range(18, 55, ErrorMessage = "Tuổi phải từ 18-55")]  
public int Age { get; set; }
```

- Thuộc tính Birthday nằm trong khoảng 01/01/1980 – 31/12/2022:

```
[Range(typeof(DateTime), "1/1/1980", "31/12/2022")]  
public DateTime Birthday { get; set; }
```

f) Chú thích [DataType]

Chú thích [DataType] quy định kiểu dữ liệu cho thuộc tính. Khi dùng Tag Helper sẽ tạo ra thẻ `<input>` có kiểu phù hợp với kiểu dữ liệu được quy định bởi chú thích này.

Cú pháp: [DataType(DataType type, ErrorMessage = string?)], trong đó tham số type là kiểu dữ liệu.

Các DataType thông dụng được chú thích này hỗ trợ: Date, DateTime, EmailAddress, ImageUrl, MultilineText, PhoneNumber, Text, Time, Upload.

Ví dụ:

- Thuộc tính Birthday chỉ nhập ngày, không nhập giờ:

```
[DataType(DataType.Date)]  
public DateTime Birthday { get; set; }
```

Ngày sinh

- Thuộc tính Password:

```
[DataType(DataType.Password)]  
public string Password { get; set; }
```

Mật khẩu

g) Chú thích [RegularExpression]

Chú thích [RegularExpression] quy định chuỗi **Regular Expression** (RegEx) dùng để kiểm tra định dạng dữ liệu cho thuộc tính.

Cú pháp: `[RegularExpression(string regex, ErrorMessage = string?)]`, trong đó tham số `regex` là chuỗi RegEx.

Ví dụ: Thuộc tính `Phone` theo chuẩn SĐT di động của VN (10 chữ số, bắt đầu bằng số 0):

```
[RegularExpression(@"0\d{9}", ErrorMessage = "SĐT không hợp lệ")]
public string Phone { get; set; }
```

h) Tham số `ErrorMessage`

Tham số `ErrorMessage` có thể nhận vào 1 tham số là `{0}` mang giá trị là tên hiển thị (`DisplayName`) của thuộc tính hiện tại. Do đó, nếu sau này chúng ta thay đổi `DisplayName` của thuộc tính thì cũng không phải sửa lại các thông báo lỗi.

Ví dụ:

```
[DisplayName("Tên đăng nhập")]
[Required(ErrorMessage = "{0} không được bỏ trống")]
public string Username { get; set; }
```

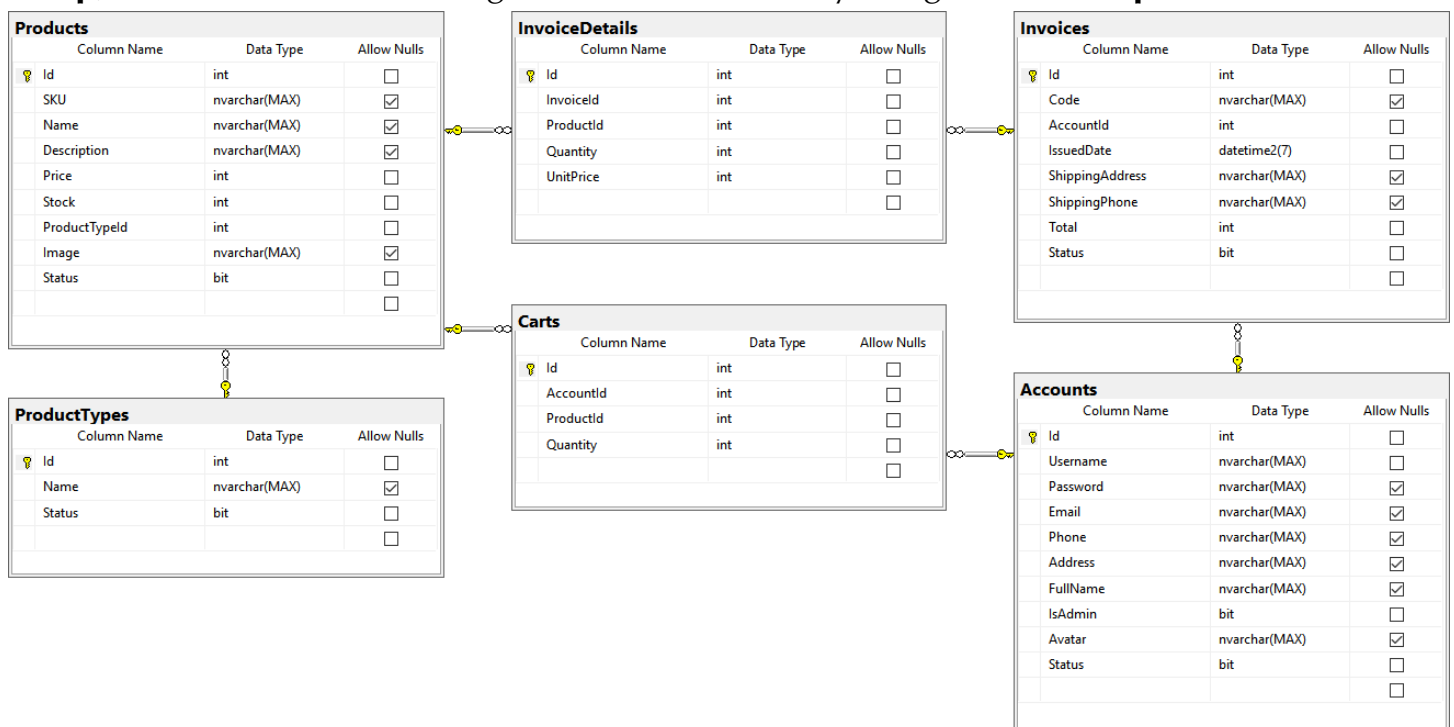
Tên đăng nhập

Tên đăng nhập không được bỏ trống

Lưu ý: Các thuộc tính kiểm tra (ngoại trừ `[MinLength]`, `[MaxLength]` và `[StringLength]`) chỉ có tác dụng trên view, không ảnh hưởng đến CSDL, do đó không cần phải thực hiện lại thao tác migration.

VII. Xây dựng CSDL Eshop

Trong môn học này, chúng ta sẽ xây dựng 1 trang web thương mại điện tử có tên là **Eshop**, và việc đầu tiên là dùng EF Code-First để xây dựng CSDL **Eshop** theo lược đồ sau:

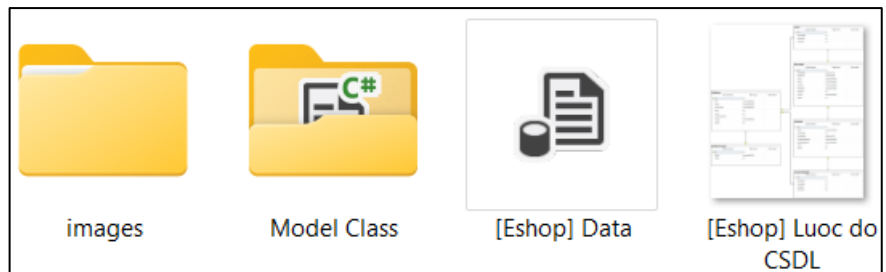


Ràng buộc và điều kiện cho các class như sau:

Tên class	Mô tả ràng buộc
Account	<ul style="list-style-type: none"> – Username và Password là bắt buộc và có độ dài 6-20 ký tự – Phone là SĐT theo chuẩn VN (gồm 10 chữ số, bắt đầu là số 0)
Product	<ul style="list-style-type: none"> – SKU (mã sản phẩm) và Name (tên sản phẩm) là bắt buộc – Price định dạng tiền tệ theo chuẩn VN (ví dụ: 45,000). Gợi ý: dùng chú thích [DisplayFormat(DataFormatString = "{0:n0}")]
ProductType	<ul style="list-style-type: none"> – Name (tên loại sản phẩm) là bắt buộc
Invoice	<ul style="list-style-type: none"> – IssuedDate (ngày lập hóa đơn) có kiểu dữ liệu là DateTime – Total định dạng tiền tệ theo chuẩn VN

Kể từ buổi 4, ta dùng CSDL Eshop gửi kèm, gồm có:

- Thư mục **Model Class** chứa mã nguồn của 6 class tương ứng với 6 bảng ở



trên, kèm theo Data Annotation cho các thuộc tính

- Thư mục **images** chứa hình ảnh sử dụng trong quá trình xây dựng trang web (ảnh đại diện cho tài khoản, ảnh minh họa cho sản phẩm). Thư mục images này cần được đặt vào thư mục gốc của ứng dụng (là thư mục wwwroot).
- File **[Eshop] Data.sql** chứa code SQL để phát sinh dữ liệu mẫu cho CSDL.
- File **[Eshop] Luoc do CSDL.png** là lược đồ CSDL Eshop.

Cách sử dụng CSDL Eshop này kể từ buổi 4:

- Thêm 6 model class vào thư mục **Models** của project.
- Khai báo 6 class này vào **EshopContext**, tạo migration và cập nhật vào CSDL để tạo CSDL Eshop.
- Mở file **[Eshop] Data.sql** bằng **SQL Server Management Studio** và chạy đoạn code trong đó để phát sinh dữ liệu.

Project **ASPNETCore_Buoi03/Eshop** đã có sẵn 6 model class nói trên và migration. Chỉ cần chạy lệnh **Update-Database** để tạo ra CSDL.