Buổi 3
Entity Framework và Code-First

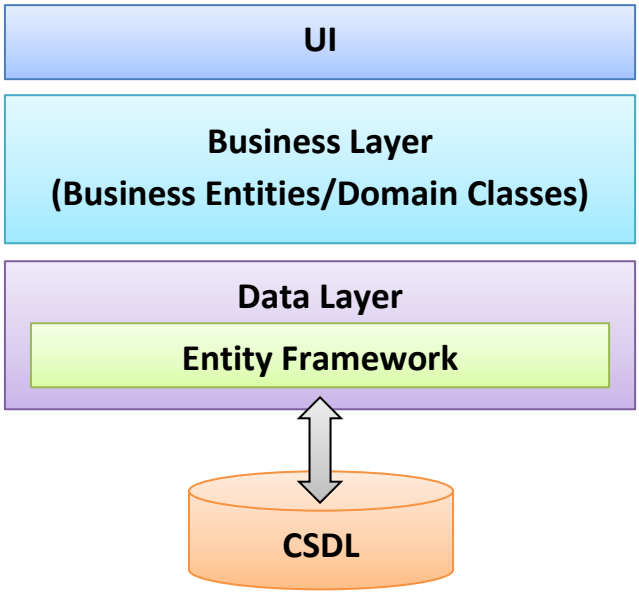
I. Entity Framework

1. Giới thiệu

Trước phiên bản .NET Framework 3.5, để tương tác với CSDL, các ứng dụng thường sử dụng ADO.NET để lấy và cập nhật dữ liệu. Việc này phải trải qua các bước: mở kết nối đến CSDL, viết truy vấn, tạo `DataTable` để lấy hoặc cập nhật dữ liệu, chuyển đổi dữ liệu kiểu object từ `DataTable` thành các đối tượng thuộc các class đã tạo (ví dụ DTO), đóng kết nối… Quá trình này khá rườm rà và dễ xảy ra lỗi.

Kể từ .NET Framework 3.5, Microsoft đã phát hành *Entity Framework*[1] để tự động hóa các thao tác liên quan đến CSDL. EF là 1 framework ORM giúp các lập trình viên .NET làm việc với CSDL đơn giản hơn bằng cách giảm bớt việc phải viết các đoạn code truy cập dữ liệu. Việc lấy và cập nhật dữ liệu, chuyển đổi và lưu trữ dữ liệu trong các class .NET sẽ được EF thực hiện tự động.

Thông thường, EF sẽ được đặt ở Data Layer:



2. Entity Framework Core

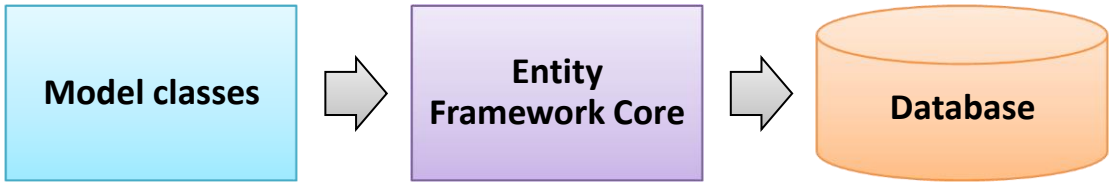
Kể từ phiên bản EF 6, EF Core được phát hành. Đây là 1 phiên bản mã nguồn mở, gọn nhẹ, đa nền tảng, thường được dùng trong các ứng dụng .NET Core, tuy nhiên nó vẫn có thể được dùng trong các ứng dụng .NET Framework 4.5 trở lên.

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…).

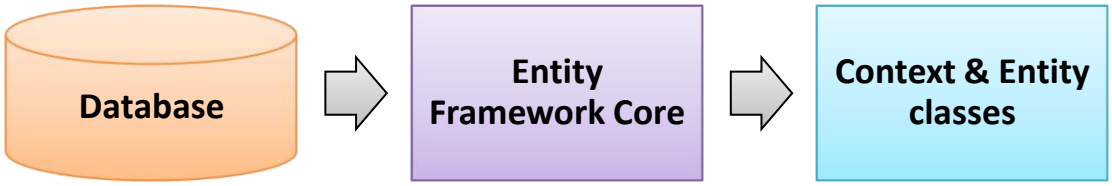
EF Core sử dụng 2 hướng tiếp cận là *Code-First* và *Database-First*, tuy nhiên EF Core chủ yếu hỗ trợ Code-First và hỗ trợ rất ít cho hướng còn lại:

- Code-First: Tạo các lớp đối tượng (*model classes*) 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[2]:

[1] Từ nay viết tắt là EF
[2] Xem phần II.2.



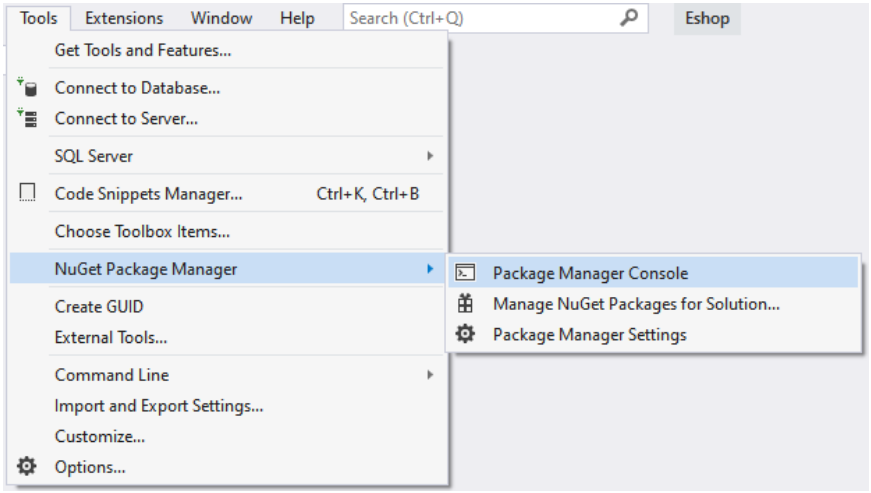
– Database-First: Tạo CSDL trước, sau đó EF Core sẽ phát sinh các lớp đối tượng (model) dựa trên cấu trúc CSDL này:



Để sử dụng được EF Core, cần phải cài đặt 1 NuGet Package đóng vai trò Database Provider kết nối đến CSDL SQL Server theo 1 trong 2 cách như sau:

– Cách 1:

+ Vào menu *Tools* → *NuGet Package Manager* → *Package Manager Console* (*PMC*):



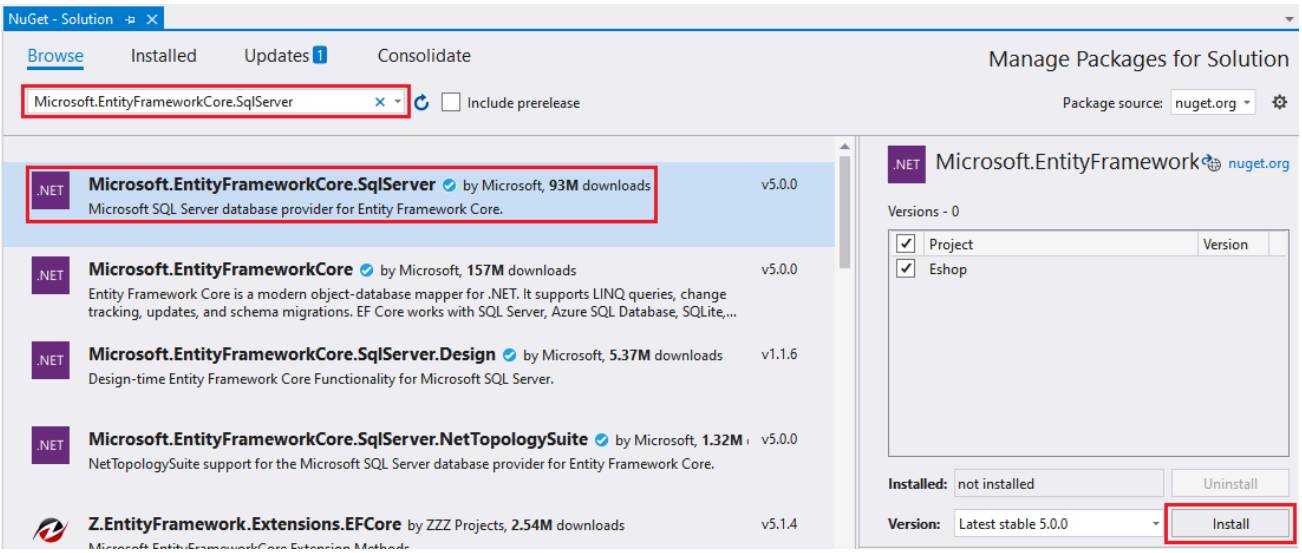
+ Trong cửa sổ *Package Manager Console*, gõ câu lệnh sau, sau đó nhấn phím *Enter*:

```
Install-Package Microsoft.EntityFrameworkCore.SqlServer
```

– Cách 2:

+ Click chuột phải vào solution hoặc project, chọn mục *Manage NuGet Packages…*
+ Trong cửa sổ hiện ra, chọn mục Browse, tìm với từ khóa sau và cài đặt:

Microsoft.EntityFrameworkCore.SqlServer



Sau đó, cài đặt package *Microsoft.EntityFrameworkCore.Tools* theo cách tương tự.

Lưu ý: Việc cài đặt các NuGet package cần phải có kết nối Internet. Ngoài ra, NuGet Package chỉ được cài đặt với solution hoặc project hiện tại. Nếu sau này muốn sử dụng NuGet Package đó cho project khác thì cần phải cài đặt lại.

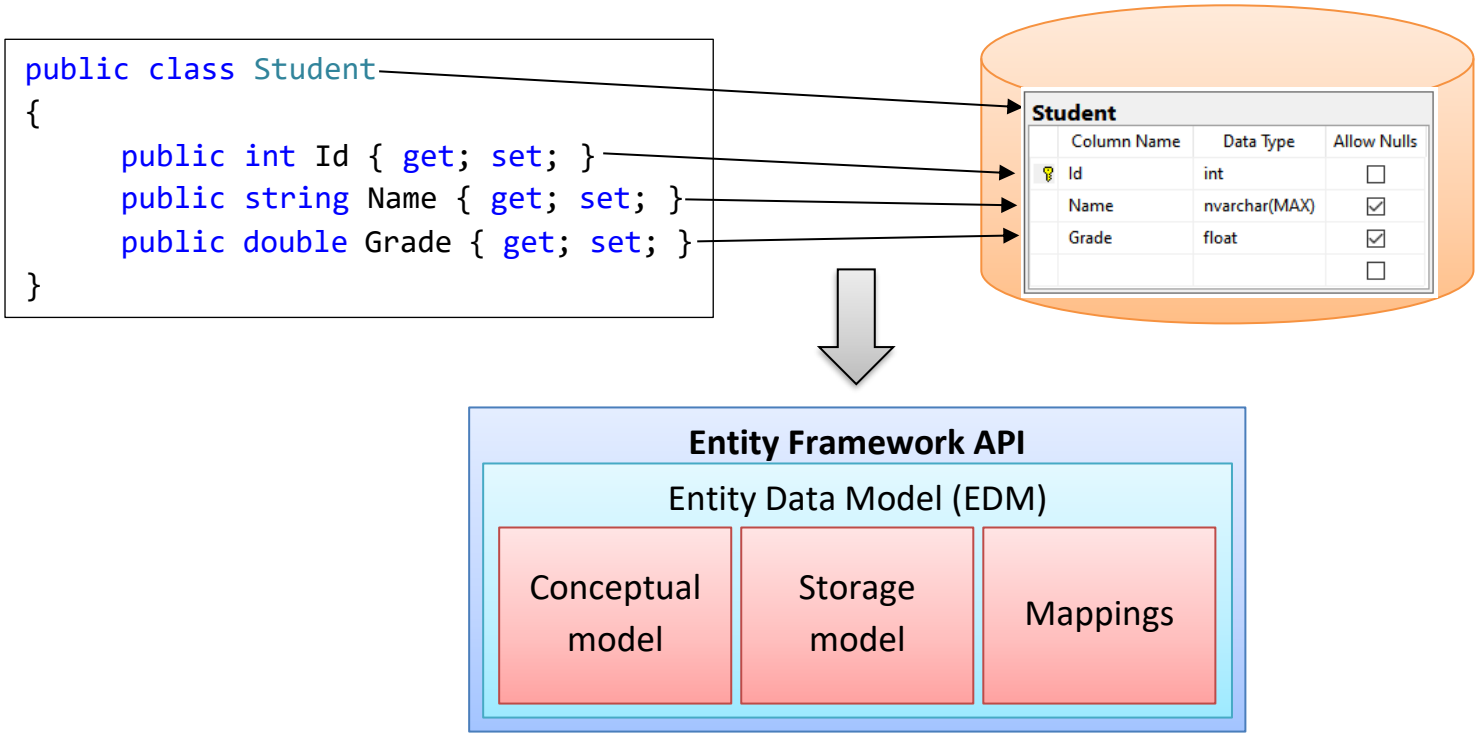
3. Cơ chế hoạt động

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.
- Dịch các câu truy vấn *LINQ*[3] 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.

Công việc đầu tiên của EF API là xây dựng 1 *Entity Data Model* (*EDM*) thể hiện toàn bộ dữ liệu:

- *Conceptual model*: EF xây dựng 1 conceptual model từ các class.
- *Storage model*: EF xây dựng 1 storage model cho CSDL. Với hướng Code-First, storage model sẽ dựa trên conceptual model. Với hướng Database-First, storage model sẽ dựa trên CSDL.
- *Mappings*: EF tạo những thông tin thể hiện ánh xạ giữa 2 loại model nói trên.



Để sử dụng được EF tương tác được với CSDL, project cần phải có 1 class kế thừa từ `DbContext`. Class này chính là thể hiện của toàn bộ CSDL, trong đó:

- Mỗi bảng sẽ thể hiện bằng 1 object của `DbSet<T>`.
- Mỗi dòng trong bảng sẽ thể hiện bằng 1 object của model class (gọi là thực thể - *entity*).
- Mỗi trường dữ liệu (cột trong bảng) sẽ thể hiện bằng thuộc tính của model class.

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 CRUD[4] từ code C# mà không cần viết 1 câu truy vấn SQL nào.

---

[3] LINQ sẽ được trình bày trong tài liệu buổi 6

[4] Thao tác CRUD sẽ được trình bày trong tài liệu buổi 4

II. Code-First

Đối với hướng tiếp cận Code-First, lập trình viên sẽ tập trung vào việc xây dựng các class của model 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 và thiết kế CSDL đã có). Theo cách này, lập trình viên sẽ bắt đầu bằng việc code (code first), sau đó, dựa vào các model class này, EF API sẽ tạo ra CSDL có cấu trúc giống như vậy.

Với project Eshop hiện tại, chúng ta đã có sẵn class `Account` nằm trong thư mục *Model*. Từ đây chúng ta sẽ xây dựng CSDL Eshop có 1 bảng *Accounts* với cấu trúc tương ứng.

Các bước thiết lập để xây dựng liên kết ánh xạ giữa model class và CSDL như sau:

- Bước 1: Tạo chuỗi kết nối đến CSDL. Mở file *appsettings.json* và bổ sung chuỗi kết nối như sau, trong đó mục *Server* và *Database* lần lượt là tên máy chủ SQL và tên CSDL. Trong khuôn khổ môn học này, máy chủ được dùng là *MSSQLLocalDb*, tuy nhiên vẫn có thể đổi sang máy chủ khác như *localhost*:

```
"ConnectionStrings": {
      "Eshop":
   "Server=(localdb)\\mssqllocaldb;Database=Eshop;Trusted_Connection=
   True;MultipleActiveResultSets=true"
}
```

- Bước 2: Tạo thư mục *Data*, trong đó tạo class `EshopContext` kế thừa từ class `DbContext` và sẽ đóng vai trò cầu nối trung gian giữa CSDL và ứng dụng web ASP.NET Core. Mỗi model class dùng để tạo ra bảng trong CSDL cần phải được khai báo dưới dạng 1 thuộc tính của class `EshopContext`, trong đó tên thuộc tính là tên bảng muốn tạo, kiểu dữ liệu của thuộc tính là `DbSet` của model class. Theo quy chuẩn của EF, tên bảng thường là danh từ số nhiều của tên class.

```
using ……
using Microsoft.EntityFrameworkCore;
using Eshop.Models;

namespace Eshop.Data
{
      public class EshopContext: DbContext
      {
            public EshopContext(DbContextOptions<EshopContext> options)
         : base(options)
            {
            }

            public DbSet<Account> Accounts { get; set; }
      }
}
```

Tên bảng muốn tạo

Tên model class

Lưu ý: `EshopContext` chỉ liên kết các class và bảng có khai báo dưới dạng thuộc tính của nó. Do đó, nếu có bổ sung thêm model class và muốn tạo bảng tương ứng, cần nhớ bổ sung `DbSet` tương ứng vào class `EshopContext`.

- Bước 3: Bổ sung câu lệnh sau vào phương thức `ConfigureServices()` của class `Startup` (file *Startup.cs*). Câu lệnh này để thêm 1 DbContext vào ứng dụng (chính là class `EshopContext` ở trên), và quy định DbContext này sẽ sử dụng chuỗi kết nối có tên là *Eshop* từ file *appsettings.json*:

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

Việc xây dựng CSDL và cập nhật những thay đổi từ model vào CSDL sẽ được giải quyết bằng *Migration* và class `EshopContext` ở trên (từ giờ sẽ gọi tắt là *context*).

III. Migration

1. Khái niệm

Trước đây, EF Code-First sử dụng các bộ khởi tạo sau để đảm bảo tạo ra được CSDL cập nhật đúng 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.

Nhược điểm của các bộ khởi tạo nói trên là chúng luôn xóa CSDL cũ và tạo lại CSDL mới, khiến cho các dữ liệu, trigger, function, stored procedure… của CSDL cũ đều bị mất đi. Do đó, các bộ khởi tạo chỉ dùng khi tạo mới CSDL, không dùng để cập nhật CSDL.

EF giới thiệu công cụ *Migration* giúp tự động cập nhật cấu trúc CSDL nếu model thay đổi mà không làm mất dữ liệu và các thông tin trong đó. Migration sử dụng một bộ khởi tạo mới có tên là `MigrateDatabaseToLatestVersion` (cập nhật CSDL lên phiên bản mới nhất).

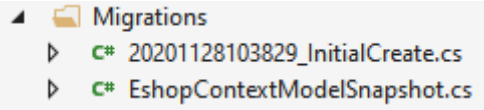
2. Khởi tạo Migration lần đầu

Để tạo migration, gõ câu lệnh sau vào PMC:

```
Add-Migration InitialCreate
```

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

Sau khi thực hiện câu lệnh này, trong project xuất hiện thêm thư mục *Migrations*:



Trong thư mục này chứa 2 file sau:

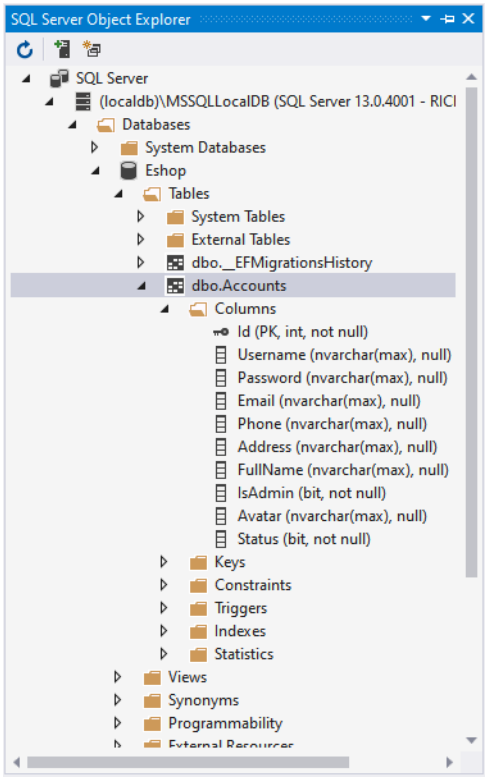
- *<timestamp>_InitialCreate.cs*, với *<timestamp>* là ngày giờ tạo ra file này. Trong file có chứa class `InitialCreate`, gồm 2 phương thức `Up()` dùng để cập nhật CSDL lên phiên bản mới nhất, và `Down()` dùng để hoàn trả (*undo*) những thay đổi.
- *EshopContextModelSnapshot.cs*: Chứa class cùng tên, là tổng thể cấu trúc của model class tính đến lần migration hiện tại (gọi là *model snapshot*).

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

```
Update-Database
```

Để xem CSDL được tạo ra bằng migration, có thể sử dụng *SQL Management Studio* hoặc dùng thanh công cụ *SQL Server Object Explorer*[5]. Với chuỗi kết nối hiện tại, CSDL Eshop sẽ được tạo ra trong *LocalDB*. Và do context mới chỉ định nghĩa `DbSet` cho class `Account` nên CSDL Eshop cũng chỉ có 1 bảng *Accounts* (xem ví dụ *3.1*):

[5] Nếu thanh công cụ này chưa mở, vào menu View → SQL Server Object Explorer



Trong CSDL được tạo ra, luôn có 1 bảng *__EFMigrationHistory* lưu lại lịch sử những lần migration từ model vào CSDL. Dựa vào bảng này, VS có thể biết được lần migration gần nhất vào CSDL là lần nào, từ đó biết được hiện tại còn thiếu những lần migration nào.

3. Cập nhật Migration khi model thay đổi

Qua thời gian ứng dụng phát triển, model class có thể có sự thay đổi. Để CSDL luôn thể hiện đúng với model, chúng ta cần thường xuyên cập nhật lại migration theo các bước sau:

- Chạy câu lệnh `Add-Migration <tên migration>` trong PMC. Do VS đã tự động thêm thời gian tạo migration nên thông thường tên của migration chỉ cần mô tả ngắn gọn thay đổi trong lần cập nhật này, ví dụ:

```
Add-Migration AddModelProduct
Add-Migration EditModelProduct_AddExpiringDate
……
```

- Sau khi chạy câu lệnh trên, VS sẽ tự so sánh với cấu trúc hiện tại của model snapshot để đối chiếu thay đổi, sau đó tạo ra 1 migration phiên bản mới.
- Chạy câu lệnh `Update-Database` trong PMC để cập nhật thay đổi từ migration vào CSDL. VS dựa vào lần cập nhật migration gần nhất (lưu trong bảng *__EFMigrationHistory* như đã trình bày ở trên) để cập nhật những lần migration còn thiếu theo thứ tự từ cũ nhất đến mới nhất.

Ví dụ: Xem ví dụ *3.2*. Bổ sung thuộc tính vào 1 model class:

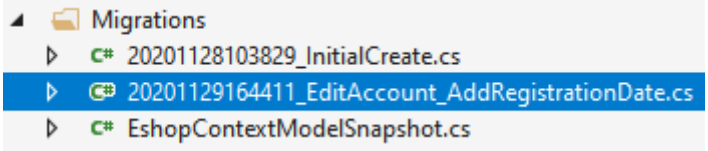
- Thêm thuộc tính `RegisterDate` (ngày đăng ký) vào class `Account`:

```
public class Account
{
    ……
    public DateTime RegistrationDate { get; set; }
}
```

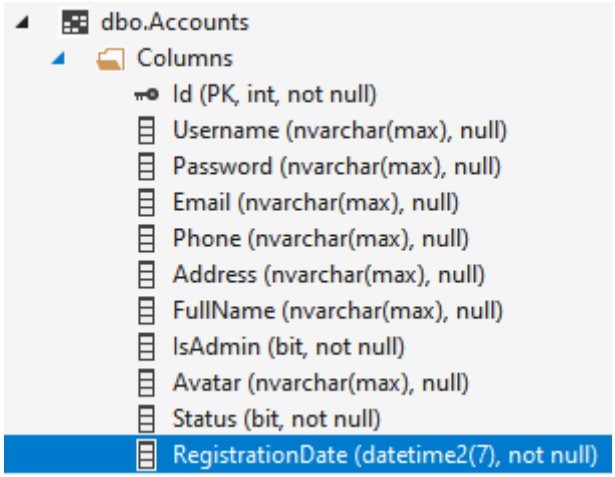
- Lần lượt chạy 2 câu lệnh sau trong PMC:

```
Add-Migration EditAccount_AddRegistrationDate
Update-Database
```

- Sau khi chạy xong câu lệnh `Add-Migration`, 1 file migration mới được tạo ra:



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

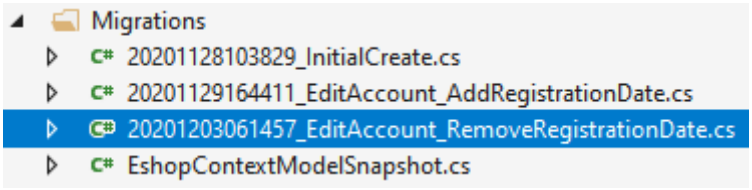


Ví dụ: Xem ví dụ *3.3*. Xóa thuộc tính khỏi 1 model class:

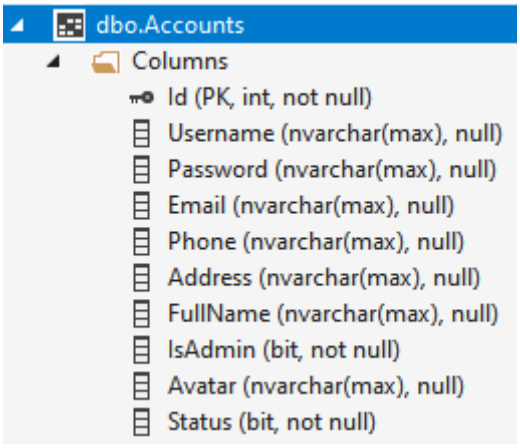
- Giả sử đã thực hiện xong ví dụ *3.2* ở trên. Sau đó, chúng ta xóa thuộc tính `RegisterDate` khỏi class `Account`.
- Lần lượt chạy 2 câu lệnh sau trong PMC:

```
Add-Migration EditAccount_RemoveRegistrationDate
Update-Database
```

- Sau khi chạy xong câu lệnh `Add-Migration`, 1 file migration mới được tạo ra:



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



Lưu ý: EF sẽ dùng migration để ánh xạ các model class sang bảng trong CSDL **khi và chỉ khi** trong context có thuộc tính `DbSet` tương ứng.

IV. Các quy tắc của Entity Framework

Việc sử dụng EF cần tuân thủ một số quy tắc mặc định sau:

1. Khóa chính

Đa số các thực thể (*entity*) trong EF chỉ có 1 khóa dùng để phân biệt các đối tượng khác nhau của thực thể đó. Do đó, khi ánh xạ sang bảng trong CSDL, khóa này sẽ trở thành khóa chính (*primary key*) của bảng.

EF quy định thuộc tính khóa có tên là `Id` hoặc là `<tên class>Id`. Nếu như muốn thuộc tính khóa có tên khác với quy tắc trên, cần sử dụng *Data Annotation*[6].

Ví dụ:

- Class `Product` có khóa là `Id`:

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

- Class `Invoice` có khóa là `InvoiceId`:

```
public class Invoice
{
    public int InvoiceId { get; set; }
}
```

- Class `Account` có khóa là `Username` (sử dụng Data Annotation):

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

2. Property và cột trong bảng

Mặc định EF sẽ ánh xạ model class thành CSDL theo quy tắc sau:

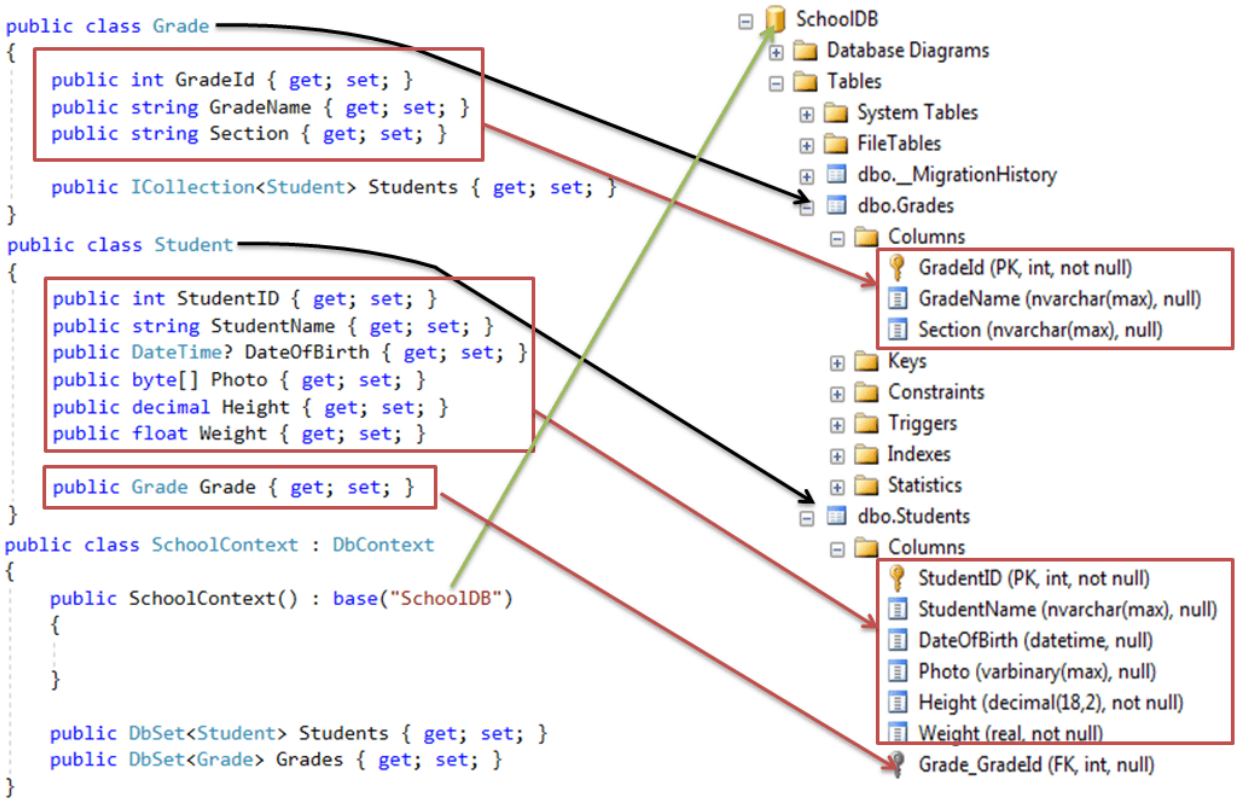
- Tên class là danh từ số ít, tên bảng là danh từ số nhiều tương ứng.
- Tên property của class được sử dụng làm tên cột của bảng.
- Thứ tự của property cũng chính là thứ tự cột.
- Kiểu dữ liệu của property được ánh xạ thành kiểu dữ liệu của SQL Server theo bảng ở phần 3 dưới đây.
- Nếu property thuộc kiểu *Reference Type*[7] (tham chiếu) hoặc kiểu *Nullable* (dữ liệu có thể mang giá trị null), cột tương ứng của nó được đánh dấu *Allow Nulls* (cho phép để trống, không chứa giá trị). Nếu property thuộc kiểu *Value Type*, cột tương ứng sẽ là *Not Null* (bắt buộc phải chứa giá trị).

Ví dụ: Ảnh minh họa cơ chế ánh xạ mặc định theo quy ước trên:

---

[6] Data Annotation sẽ được trình bày ở tài liệu buổi 4

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



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 tắc 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` (mặc định) `date, datetime, smalldatetime`
`String, string`	`nvarchar(MAX)` (mặc định) `varchar, char, nchar`

4. Null và Not Null

SQL Server cho phép một cột bất kỳ có thể nhận giá trị null, với ý nghĩa rằng ô tương ứng có thể không chứa giá trị. Ví dụ, một cột số nguyên (`int`) nếu được đánh dấu *Allow Nulls*, ô tương ứng của nó có thể không chứa giá trị.

Tuy nhiên, EF lại quyết định xem một cột có thể nhận giá trị null hay không dựa vào kiểu dữ liệu của property. Các kiểu dữ liệu của .NET được chia làm 2 loại chính: Value Type và Reference Type. Enum, struct thuộc Value Type còn class, interface, delegate thuộc Reference Type.

Đối với các kiểu Reference Type (ví dụ: `string`, class tự định nghĩa…), biến của chúng có thể nhận giá trị null. Do đó, cột tương ứng của bảng tự động được đánh dấu *Allow Nulls*.

Đối với kiểu Value Type (ví dụ: `int`, `bool`, `double`, `DateTime`,…), biến của chúng không thể nhận giá trị null. Do đó, cột tương ứng của bảng không được đánh dấu *Allow Nulls*.

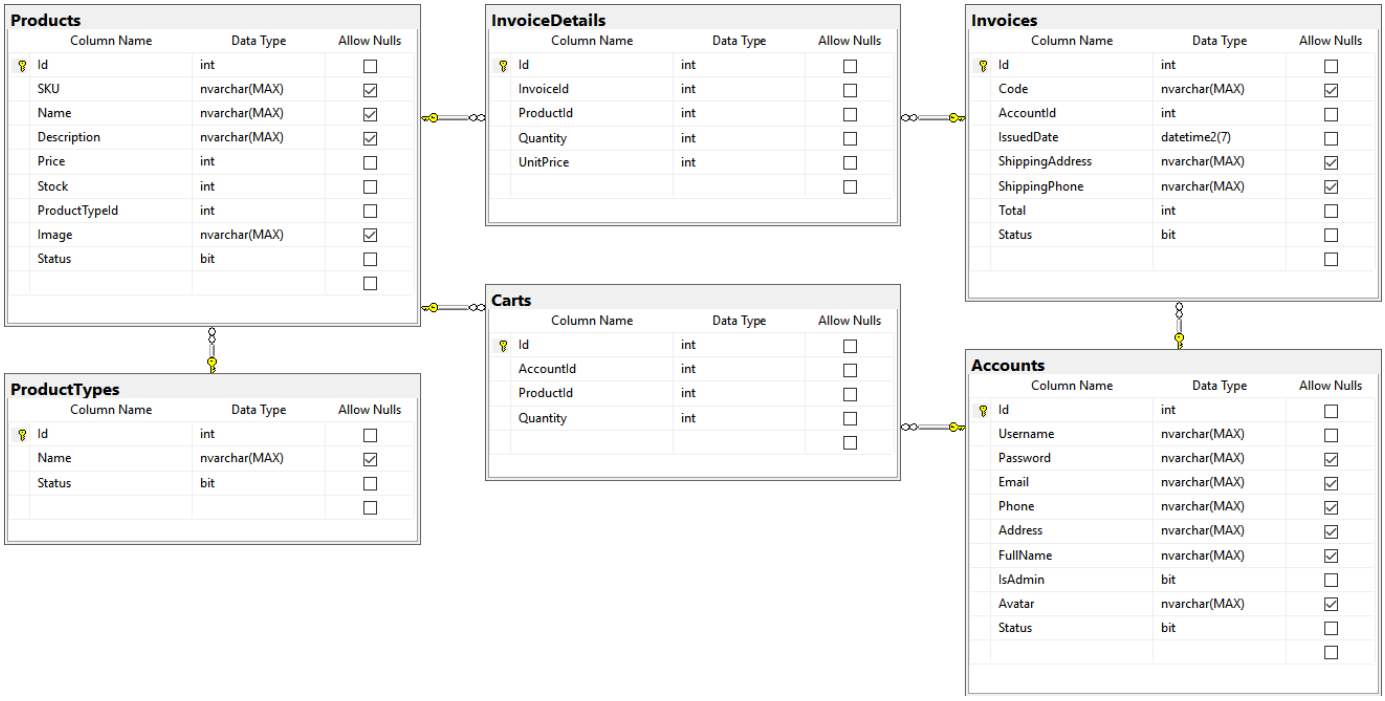
Khi tạo model class, bạn có thể cho phép một cột tương ứng của property có thể nhận giá trị null bằng cách đặt thêm dấu ? phía sau tên kiểu dữ liệu. Cách viết này yêu cầu EF đánh dấu cột tương ứng trong bảng là *Allow Nulls*.

Ví dụ:

```
public class Account
{
    public int? Age { get; set; }          // Có thể null
    public bool? Status { get; set; }      // Có thể null
}
```

V. Xây dựng CSDL Eshop

Trong môn học này, chúng ta sẽ xây dựng 1 trang web thương mại điện tử có tên là Eshop. Lược đồ CSDL của trang web này như sau:



SV cần tự viết các model theo quy tắc của Entity Framework, sau đó sử dụng Code-First và Migration để tạo ra CSDL như lược đồ trên (trừ khóa ngoại).

VI. Bài tập

Phân tích đặc tả chức năng dưới đây và thiết kế lược đồ CSDL cho trang web thương mại điện tử WannaBuy. Sau đó, với project *WannaBuy*, sử dụng Entity Framework và Code-First để xây dựng các model tương ứng. Dùng Migration để phát sinh ra CSDL như lược đồ vừa thiết kế.

Lưu ý: Ở thời điểm hiện tại, chúng ta có thể phân tích khóa ngoại giữa các bảng trong CSDL, nhưng sẽ không xây dựng khóa ngoại giữa các model.

Đặc tả chức năng của 1 trang web thương mại điện tử *WannaBuy* như sau:

1. Sản phẩm (*product*) mà trang web này kinh doanh là điện thoại di động, gồm có các thông tin sau: Tên điện thoại, Tên hãng sản xuất, Cấu hình, Giá tiền, Số lượng tồn kho, Ảnh minh họa.
2. Người dùng (*user*) của trang web gồm có:
    a. Khách vãng lai (*guest*): người dùng chưa có tài khoản trong hệ thống, hoặc đã có tài khoản nhưng không đăng nhập vào hệ thống.
    b. Khách hàng (*customer*): người dùng có tài khoản và đã đăng nhập vào hệ thống.
    c. Quản trị viên (*administrator*): người dùng có toàn quyền đối với hệ thống.
3. Thông tin 1 tài khoản (*account*) của người dùng gồm có: Tên đăng nhập, Mật khẩu, Email, Họ tên, Địa chỉ, SĐT, Ảnh đại diện.

4. Tất cả người dùng đều có thể:
    a. Xem danh sách sản phẩm.
    b. Xem danh sách sản phẩm của 1 hãng sản xuất nào đó.
    c. Xem chi tiết 1 sản phẩm.
    d. Tìm kiếm sản phẩm theo tên, theo hãng sản xuất, theo 1 khoảng giá tiền.
5. Khách vãng lai có thể đăng ký tài khoản để trở thành khách hàng.
6. Khách hàng có thêm các chức năng sau:
    a. Thêm 1 sản phẩm vào giỏ hàng (*cart*).
    b. Xóa 1 sản phẩm khỏi giỏ hàng hoặc xóa toàn bộ giỏ hàng.
    c. Thanh toán.
    d. Xem danh sách các hóa đơn (*invoice*) mình đã mua.
    e. Thay đổi thông tin cá nhân (ngoại trừ Tên đăng nhập).
    f. Đăng nhập, Đăng xuất. Khi khách hàng đăng xuất, trạng thái giỏ hàng của khách hàng đó được giữ nguyên.
7. Quản trị viên có tất cả chức năng của khách hàng và thêm các chức năng sau:
    a. Quản lý (Xem, thêm, sửa, xóa) tài khoản.
    b. Quản lý sản phẩm.
    c. Quản lý danh sách nhà sản xuất.
    d. Quản lý hóa đơn.