# Configuring Nonrelational Properties in EF Core

## EF Core Configuration By Convention

Tức là EF Core sẽ dùng chính tên và kiểu dữ liệu khai báo của property trong Model config luôn db theo như vậy. Nếu ta muốn override nó thì có 2 cách là sử dụng Data Annotations hoặc Fluent API

Theo convention. Property có tên “Id” sẽ được dịch thành primary key trong db. Hoặc nó được combine bởi <Name of class>+Id, ví dụ như StudentId

Khi sử dụng config bởi convention, giá trị nullable của một cột dựa trên kiểu dữ liệu từ model class của chúng ta. VÍ dụ như “Name” property có thể có một giá trị “Null” (bởi vì mặc định của một kiểu string là null) nhưng Age thì là không thể (Age có kiểu là type int).

Nếu chúng ta muốn Age property nullable trong db, chúng ta có thể sử dụng “?” suffix như là int? Age hoặc sử dụng generic Nullable<T> như là Nullable<int> Age

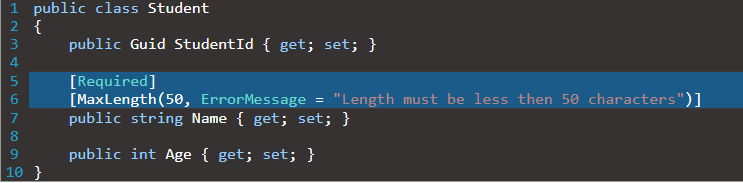
## EF Core Configuration via Data Annotations

Data Annotations là các .NET attributes, chúng được sử dụng để validate và config các tính năng của database. 2 namespace mà có liên quan ở đây là

System.ComponentModel.DataAnnotations

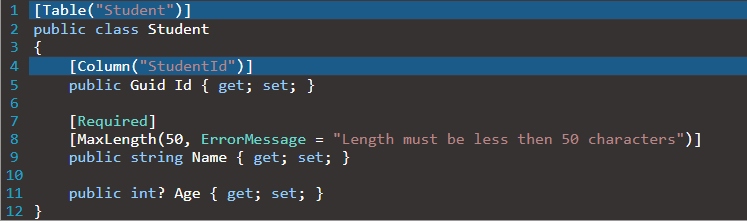
System.ComponentModel.DataAnnotations.Schema

Attributes từ namespace đầu tiên (trong 2 namesspace liệt kê bên trên) phần lớn là liên quan tới việc property validation, thằng namespace còn lại thì liên quan đến config.



Bên trên là một ví dụ. Với “Required” attribute, “Name” property sẽ không thể bị null, và với “MaxLength” thì nó sẽ giới hạn độ dài của column này trong database.

Chúng ta có thể thêm các sử đổi tới Student class bằng việc adding [Table] và [Column] attribute:



### Table attribute

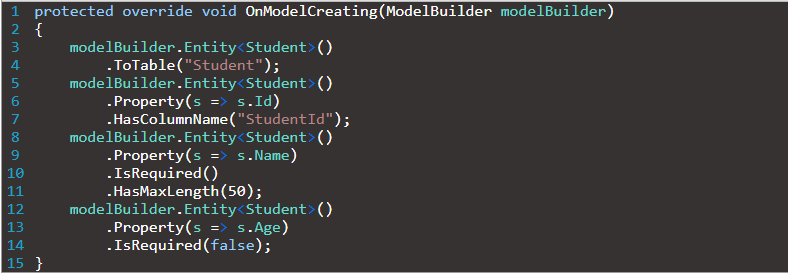
Bằng việc sử dụng [Table] attribute, Chúng ta đang hướng EF Core tới đúng table hoặc schema. Tên của table trong database là “Students” bởi vì DbSet<T> property có tên là Students trong ApplicationContext. Nhưng [Table] attribute sẽ override nó. Nếu một table chúng ta đang map tới không phải là một deffault schema, CHúng ta cần sử dụng [Table(“TableName,Schema=”SchemaName””)]

### Column attribute

[Column] attribute cung cấp EF core thông tin về column để map tới db. Chúng ta có thể cung cấp “Order” và Database Type của column với attribute [Column(“ColumnName”,Order=1,TypeName=”nvarchar(50)”)]

## Using the Fluent API Approach

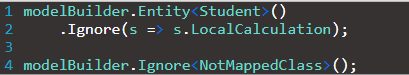
The Fluent API là một tập hợp các methods, chúng ta có thể sử dụng chúng trên ModelBuilder class (tức là các method này là của class ModelBuilder). Chúng thì có sẵn trong OnModelCreating method trong Context class của chúng ta. Cách tiếp cận này cung cấp cho chúng ta rất tuyệt vời về EF Config.



Tại thời điểm cả 3 cách tiếp cận đều được áp dụng, thằng này sẽ được gọi đầu tiên

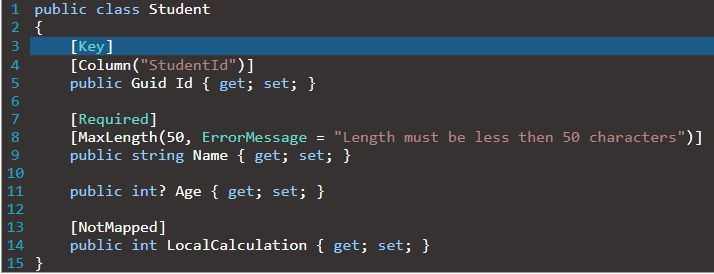
## Excluding Entities or Classes from the Mapping

EF Core sẽ map tất cả property có trong model, nếu chúng ta không muốn map một property nào đó thì chúng ta phải sử dụng [NotMapped] attribute hoặc có thể set ở trong Fluent API



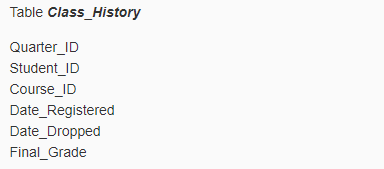
## PrimaryKey Config

Với Convention như ta đã đề cập bên trên, khi đặt tên property chúng sẽ tự detech primary key. Nếu chúng ta muốn override là trường khác làm key thì ta có thể sử dụng [Key] attribute ngay bên trên tên trường hoặc sử dụng Fluent API





Với composite key, chúng ta chỉ sử dụng Fluent API bởi vì EF Core không hỗ trợ Data Annotations

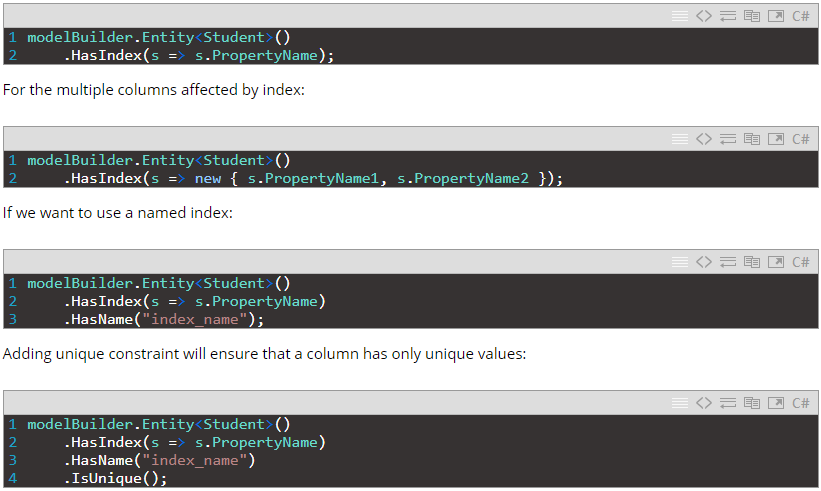


Ví dụ như trên là composite key, khi mà có 3 key là 3 trường đầu tiên. Nếu chúng đứng độc lập sẽ chưa chắc là unique nhưng nếu chúng combine lại với nhau chắc chắn sẽ là một giá trị unique

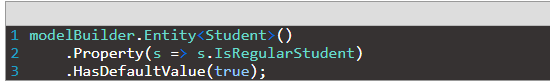
## Working with Indexes and Default Values

Chúng ta sẽ sử dụng Fluent API để add indexes tới db bởi vì chỉ có cách này làm được =))

Để add an index tới property cần, chúng ta có thể sử dụng statement như sau.



Chúng ta có thể set default value thông qua Fluent API như sau:



## Recommend

### By Convention

Chúng ta luôn bắt đầu config với convention. Vì vậy đặt tên class giống tên table name, tên primary key phù hợp vs convention, tên các property giống tên columns sẽ là sự lựa chon ưu tiên của chúng ta.

### Data Annotations

Về validation config, như là required hoặc max length validation, ta sẽ thích sử dụng Data Annotations hơn là Fluent API

### Fluent API

Với cái này chúng ta có thể làm mọi thứ. Chúng ta phải sử dụng cái này nếu chỉ cái này hỗ trợ laoij config ấy hoặc chúng ta muốn ẩn config trên setup từ model class.

# Migrations and Seed Data with EF Core (Di cư model => database, và gieo hạt (dữ liệu khởi tạo) dữ liệu)

## Creating and Applying Migrations in EF Core

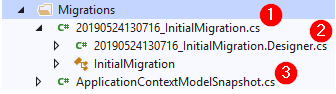
Việc sử dụng migrations là cách thức tiêu chuẩn để tạo và update một database với EF Core. Migration gồm 2 bước: Tạo Migration và Applying migration. Database scheme phải được căn chỉnh với database model và mỗi thay đổi trong database model cần được migrate tới database

Để create một migration, chúng ta có thể sử dụng VS’s Package Manager Console window hoặc command window. Với PMC window, the command this:

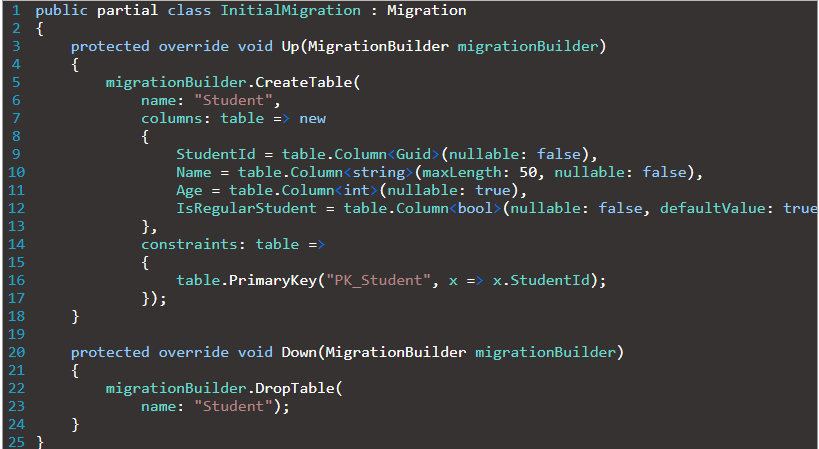
**Add-Migration MigrationName [options]**

## Actions that Take Place Behind the Scene

Sau khi chúng ta execute Add-Migration có một vài thứ đằng sau bối cảnh. Việc đầu tiên nó làm là kiểm tra các class của chúng ta, entity classes có liên quan và bất cứ config nào được áp dụng. Sau đó, nó tạo 3 files khác trong Migrations folder:



File số 3 giữ model của database và nó được update mỗi lần khi một new migration được thêm vào. File số 1 và 2 là các files chứa và mô tả migration mới đã được tạo.



File này có 2 method là up và down . Up method bao gồm các commands sẽ được executed khi chúng ta apply migration này. Down sẽ được executed khi chúng ta remove migration này.

## Applying Created Migration

Sau khi chúng ta đã tạo Migration, chúng ta có thể apply các thay đổi này tới database.

**Update-Database [options]**

Sau khi chúng ta execute câu lệnh trên, chúng ta sẽ nhìn thấy tất cả các actions khác nhau mà EF Core làm cho chúng ta để apply migration đã tạo từ trước. Và kết quả là chúng ta có Student table đã tạo với config đã được cung cấp từ khi chúng ta setting.

Có vài điều thực tế rất quan trong chúng ta nên biết về EF Core’s migrations. Nếu chúng ta nhìn vào database sẽ nhìn thấy \_EFMigrationsHistoriy EF Core sử dụng table này để theo dõi tất cả các migration đã được apply. Vì vậy nếu chúng ta tạo một migration khác trong code và apply nó, EF Core sẽ chỉ apply migration mới nhất đã tạo.

Tức là nó sẽ apply cho thằng migration mới nhất, và trong bảng nó tự tạo kia nó có lưu thằng mới nhất mà được apply đến.

**Phần Migration sẽ tạm dừng chỗ này, khi nào dùng thì đọc tiếp và viết tiếp.**

# **EF Core Relationships**

## Concepts and Navigational Properties

Chúng ta có nhiều entities, các entities có thể có các mối quan hệ với nhau. Trong một mối quan hệ thì một trong số chúng sẽ là Principal entity và thằng kia sẽ là Dependent entity. Principal entity là main entity trong relationship. Nó chứa primary key như là một property cái mà dependent entity refer tới thông qua foreign key

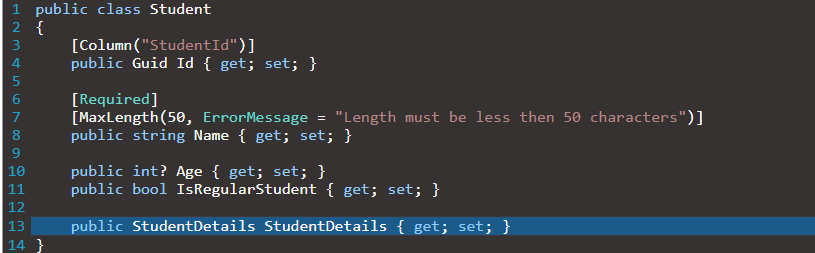
Navigational Properties là properties chứa một single class hoặc collection của classes cái mà EF Core sử dụng để link entity classes

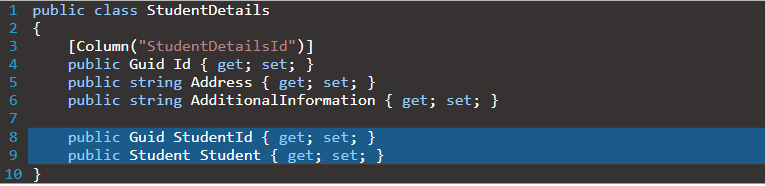
Required and Optional relationships in EF Core . The required relationship là một relationship mà foreign key không thể null. Thằng optional thì ngược lại

## Config One-to-One

The one-to-one relationship có nghĩa rằng. Một record trong một table chỉ có mối quan hệ tới một record trong bảng khác. Mối quan hệ này không phổ biến bởi vì nó thường được xử lý như là đưa tất cả dữ liệu vào một bảng, nhưng thi thoảng ta vẫn muốn tách bảng.

Cách dễ nhất để config kiểu relationship này thì chúng ta sử dụng Convention



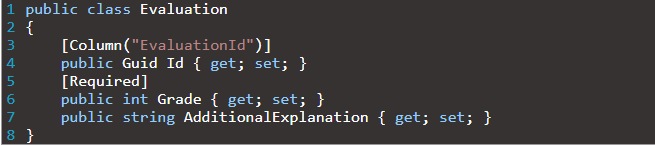


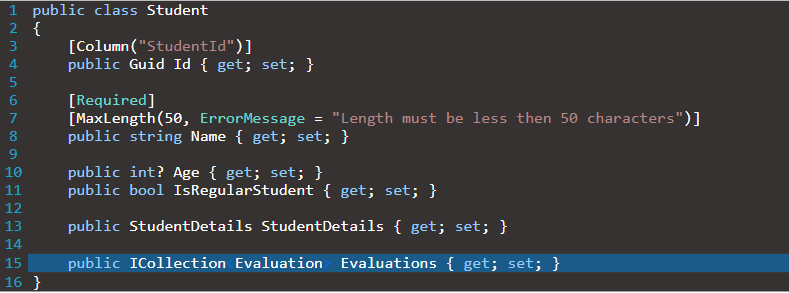
Chúng ta có thể thấy Student class có một reference navigation property đối với StudentDetails class và StudentDetails class có một foreign key và navigation property Student.

Và đối với thằng này chúng ta không cần khai báo trong Context nhưng mà nó cũng vẫn cứ được tạo trong database. Lý do tại sao

EF Core sẽ tìm kiếm tất cả public DbSet<T> properties trong DbContext class để tạo các tables trong database. Sau đó nó tìm kiếm tất cả public properties trong T class để map columns. Nhưng nó cũng tìm kiếm tất cả public navigational properties. Vì vậy trên ví dụ trên, khi tạo Student table nó sẽ tìm đến navigational property để tạo thêm một bảng là StudentDetails

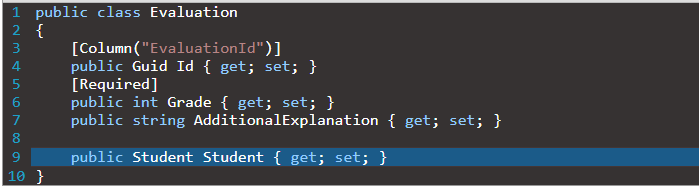
## One-to-Many Relationship





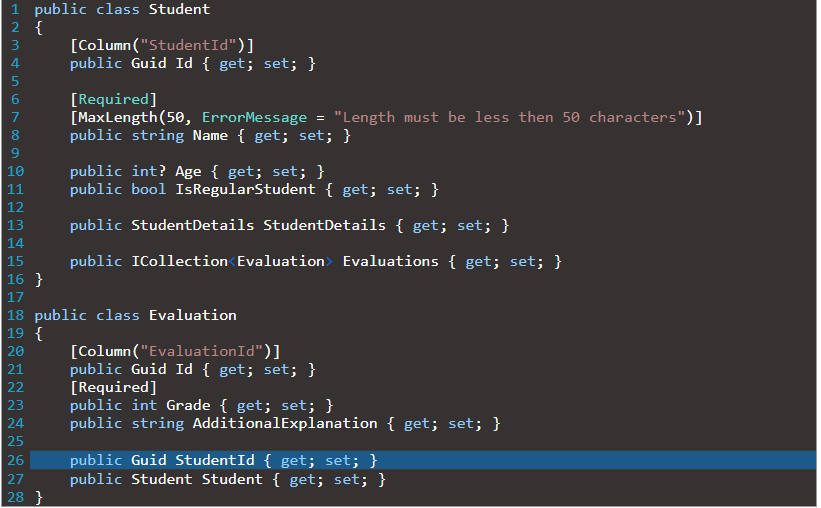
Vẫn sử dụng Convention để tạo mối quan hệ này, có nhiều cách để tạo mối quan hệ này, như bên trên là một cách khi mà Evaluation là bên nhiều, Student là bên một => có nghĩa là một student được sử dụng bởi nhiều evaluation => student giữ một danh sách các evaluation.

Cách 2: Ta sẽ để Evaluation giữ Student, vì Evaluation là nhiều, tức là bên mà nhiều record evaluation có mối quan hệ với một student và mỗi record evaluation chỉ có thể có quan hệ với một record bên student. Với cách này thì bên Student sẽ không có property Navigational



Cách 3: Với cách này bên Student ta khai báo một property Navigational là một collection của Evaluation và bên Evaluation thì khai báo một property Navigational của Student.

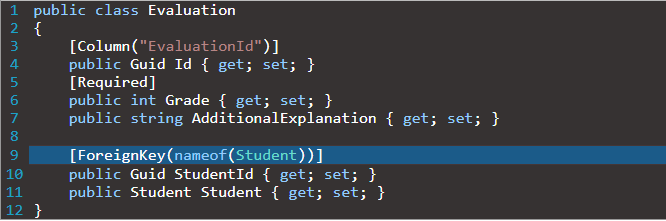
Với cả 3 cách thì Database đều tự sinh ra một foreign key là StudentId trong bảng của Evaluation. Để mà ta có thể minh bạch và set thêm các config về Required hay Optional thì ta cần tạo ra một minh bạc property StudentId trong model và config nó với Navigational property.



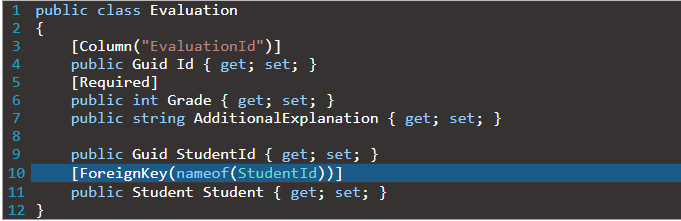
### Data Annotations

Thằng này chứa 2 attributes để thể hiện quan hệ này là [ForeignKey] và [InverseProperty] attributes.

[ForeignKey] cho phép chúng ta định nghĩa foreign key cho navigational property trong model class.



Bên trên ta đã trao cho StudentId là [ForeignKey] để thể hiện tên của navigational property Student nhưng chúng ta cũng có thể làm ngược lại với thằng navigational property như sau

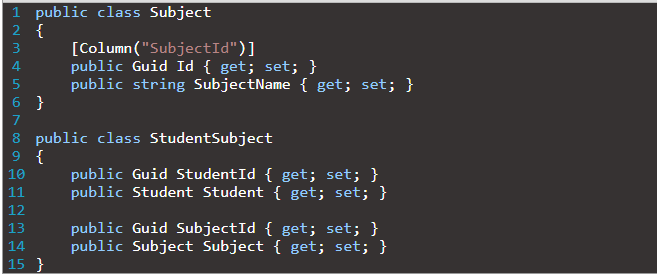


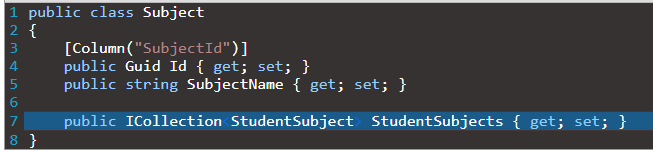
### Fluent API approach for the One-to-Many Configuration

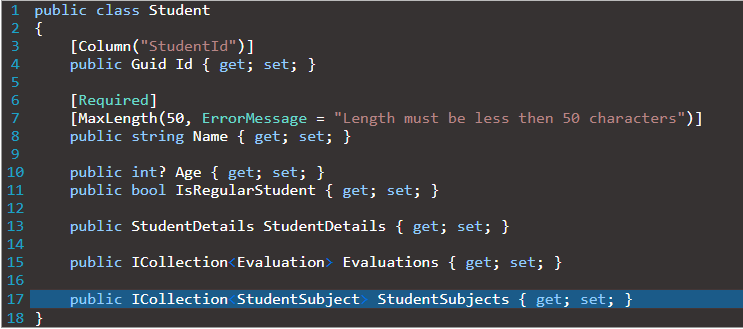
Để tạo với Fluent API chúng ta xóa attributes của khai báo Annotations và chỉnh sử StudentConfiguration như sau.



## Many-to-Many Relationship



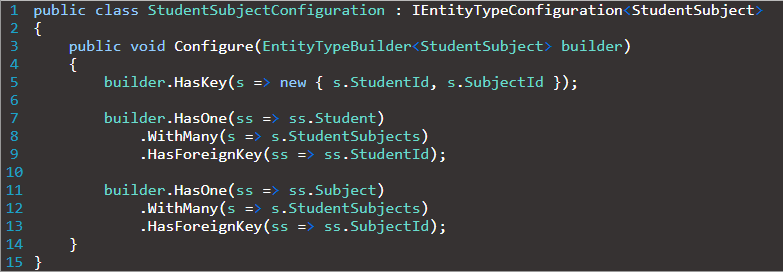


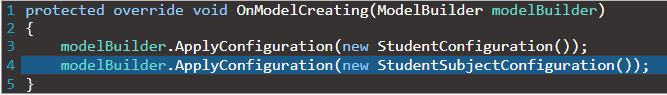


Để có mối quan hệ nhiều nhiều, ta cần tạo một bảng phụ

Bên trên là cách khai báo với Convention thông qua 2 mối quan hệ 1-n từ student tới studentsubject và subject tới studentsubject

### Khai báo với Fluent API

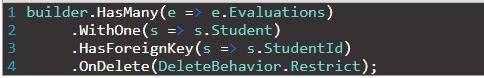




# OnDelete Method

OnDelete method config là các actions liên quan đến mối quan hệ giữa các entities. Chúng ta có thể thêm các method này tới cuối trong config mới quan hệ để quyết định cách delete actions sẽ được thực thi.

* Restrict – Chúng ta sẽ không thể delete principal entity nếu nó có các dependent entity đang tồn tại
* SetNull – The dependent entity sẽ không bị delete nhưng foreign key sẽ bị set giá trị null.
* ClientSetNull – Nếu EF Core tracks dependent entity foreign key của nó set tới null và entity không bị delete. Nếu EF Core không track the dependent entity, database rules apply
* Cascade – The dependent entity bị xóa cùng với principal entity



Để mà có thể set được cái này thì ta chỉ có thể sử dụng Fluent API

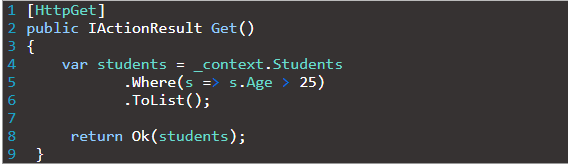
# Database Queries in EF Core

## Understanding Queries in EF Core

CHúng ta có thể bắt đầu querying data từ database bằng việc sử dụng EF Core. Mọi query bao gồm 3 phần chính:

* Connection tới database thông qua ApplicationContext’s DbSet property
* Series of LINQ and/or EF Core commands
* Query execution

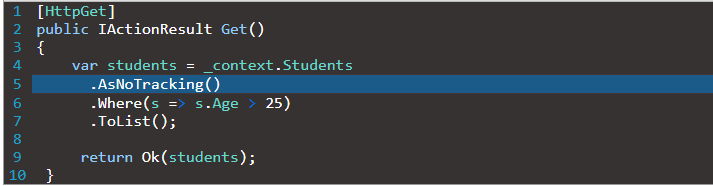
Phần thứ 2 được sử dụng thường xuyên, nhưng thi thoảng có thể bỏ qua nếu chúng ta muốn trả về tất cả các row của table



Từ ví dụ query trên, chúng ta có thể nhìn thấy tất cả các phần đã đề cập. The “\_context.Students” là phần đầu tiên, nơi mà chúng ta access tới Student table trong database thông qua DbSet<Student> Students property. The “.Where(s=>s.Age>25)” là phần thứ 2 của query nơi mà chúng ta sử dụng LINQ command để select chỉ các row mà chúng ta yêu cầu. Cuối cùng chúng ta có “ToList()” method cái mà executes this query.

TIP: Khi mà chúng ta chỉ viêt read-only queries trong EF Core, chúng ta sẽ luôn add AsNoTracking method để tăng tốc độ cho việc execution.

Chúng ta sẽ bàn về việc track này sau, bây giờ thì chỉ cần biết là EF Core không track change (khi mà chúng ta apply AsNoTracking) trên entity đã load cái này sẽ tăng tốc độ query execution:



## Different Ways of Building Relational Queries

Có vài cách tiếp cận khác nhau để lấy được dữ liệu của chúng ta:

* Eager loading
* Explicit loading
* Select (Projection) loading
* Lazy loading

## Relational Database Queries with Eager Loading in EF Core

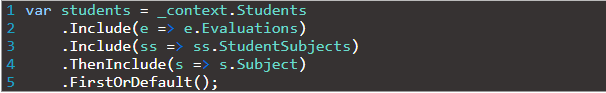
Với cách này, EF Core includes the relationships trong kết quả query. Nó sử dụng 2 method khác nhau là Include() và ThenInclude()



Điều quan trọng ở đây mà ta cần biết là chúng ta có thể include tất cả entities trong our queries via Student entity bởi vì nó có relationships với các entities khác.

### ThenInclude

Để chỉnh sửa thêm câu lệnh query cho việc including ở second-level relationship properties, chúng ta có thể attach ThenInclude method right after the Include method. Vì vậy, với Include method, chúng ta đang load the first-level relationship properties và một khi chúng ta attach ThenInclude, chúng ta có thể đi sâu hơn bên trong relationship graph.



Chúng ta có thể đi đến với độ sâu bất kì với ThenInclude method bởi vì nếu relationship không tồn tại, the query không fail và đơn giản là nó không trả về bất cư sthuws gì. Cái này cũng áp dụng với Include method

### Eager Loading Advantages and Disabdvantages and Console Warnings

Lợi thế của thằng này là chúng ta có thể includes relational data với Include và ThenInclude một cách hiệu quả, sử dụng ít database access

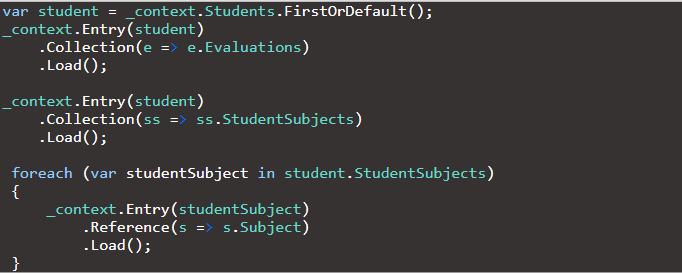
Nhược điểm là nó sẽ load tất cả dữ liệu, thâm chí cả cái chúng ta không cần

## Explicit Loading in EF Core

Với cách này, EF Core loads relationships một cách minh bạch bên trong entity đã được load. Vì vậy chúng ta hãy nhìn sự khác biệt giữa các cách tới explicityly load relationships

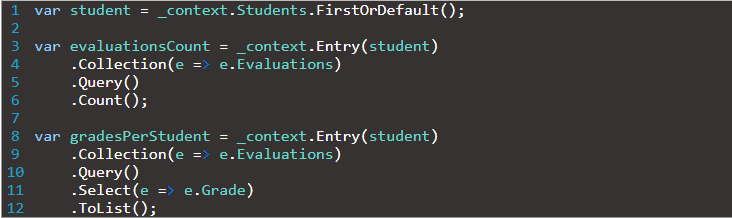
Với cách này chúng ta không còn sử dụng đển DbSet<Student> Students property nữa. Mà thay vào đó là sử dụng hàm Entry().

Khi mà chúng ta muốn load các relationship của nó, nếu là một Collection method và với single entity là navigational chúng ta sử dụng Reference method



### Queries in EF Core with Explicit Loading

Khi làm việc với Explicit, chúng ta có thêm một commad có sẵn, nó cho phép query để áp dụng tới relationship. Vì vậy thay vì sử dụng Load method như ở ví dụ trước, chúng ta sẽ sử dụng query method

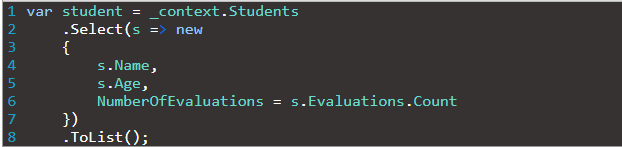


Đây chính là lợi thế của Explicit Loading, chúng ta có thể load relationship trong một entity class muộn hơn khi mà thời điểm chúng ta thật sự cần nó. Một lợi ích khác là chúng ta có thể chia việc load relationships nếu chúng ta có complex business logic. Relationship loading có thể được xóa bởi các method khác hoặc thậm chí một class. Vì vậy làm cho code dễ đọc hơn và dễ maintain

Nhược điểm là chúng ta có nhiều lần đi đến database để load tất cả các relationships mà chúng ta cần, vì vậy nó làm cho việc query kém hiệu quả hơn.

## Select (Projection) Loading

Với cách này chúng ta sử dụng Select method để chỉ chọn các properties chúng ta cần cho kết quả.



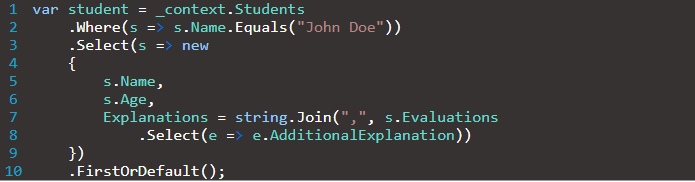
Với cách này chúng ta lên kế hoạch chỉ với dữ liệu cái mà chúng ta muốn trả về trong reponse. Dĩ nhiên nếu chúng ta không muốn trả về một đối tượng nặc danh thì chúng ta có thể tạo DTO object để trả về.

Lợi thế của cách này là chúng ta có thể chọn dữ liệu chúng ta muốn load nhưng hạn chế là chúng ta phải viết code cho mỗi property chúng ta muốn include để trả về.

## Layzy Loading in EF Core

## Client vs Server Evaluation

Tât cả các queries chúng ta đã viết xa hơn thì nó là EF Core có thể translate tới SQL commands. Nhưng EF Core có một tính năng được gọi lad Client vs Server Evaluation cái cho phép chúng ta include methods trong our query cái mà không thể được translate trong SQL command. Các command này sẽ được executed ngay khi dữ liệu được nhận từ database



Vậy thứ tự của chúng ta sẽ là . Đầu tiên cần sử dụng Eager loading để load cái ta muốn. Sau khi lấy được row rồi thì tiếp theo chúng ta có thể dùng Explicit để loading relationship nếu muốn. Hai thằng này sẽ là thằng chính để sử dụng còn các thằng khác tạm thời chưa cần quan tâm.

## Raw SQL Commands

Cái này chưa cần thiết

# Modifuing Data with EF Core

## ChangeTracker and State of the Entity in EF Core

DbContext chỉ chứa đứng 3 property: ChangeTracker, Database, and Model. Model thì đã nói ở bên trên và ở đây chúng ta sẽ bàn hơn về ChangeTracker và State properties

The ChangeTracker property cung cấp việc truy cập tới change tracking infomation and operations về entity đã load hiện hành. Khả năng này rất quan trọng khi chúng ta muốn execute any of the database modification operations. EF Core có phân loại thông tin này mỗi khi chúng ta tạo, modify hoặc delete một entity

Xa hơn, EF Core không execute bất cứ operation cho đến khi chúng ta call SaveChanges method. Vì vậy việc biết operation nào chúng ta muốn execute là quan trọng tới EF Core để gọi SaveChanges method.

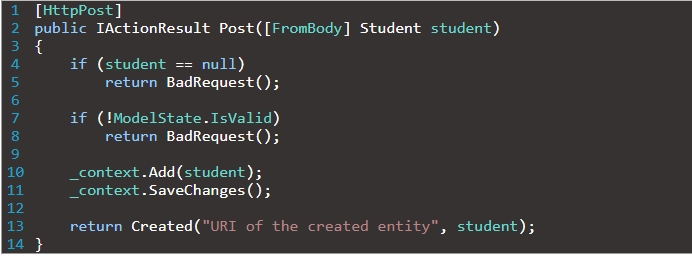
Mỗi tracked entity có State property được gắn với nó. Khi chúng ta sử dụng context object để load entity mà không gắn thêm AsNoTracking method hoặc chúng ta thay đổi state của nó thông quan Update, Remove or Add methods, entity này sẽ là tracked entity. Giá trị của State property có thể được chứa với \_context.Entry(our\_entity).State command.

Dưới đây là các trạng thái của entities tracked:

* Detached – The entity không được tracked và call SaveChanges method sẽ không có cái gì xảy ra với entity này
* UnChanged – The entity đã load từ database nhưng không có thay đổi. The SaveChanges method sẽ bỏ qua nó
* Added – The entity không chứa trong database và việc call SaveChanges method sẽ add nó tới database
* Modified – The entity tồn tại trong database và đã được chỉnh sửa, vì vậy việc call SaveChanges method sẽ chỉnh sửa nó trong database
* Deleted – The entity chứa nó trong database và ngay khi chúng ta call SaveChanges method nó sẽ bị xóa từ database

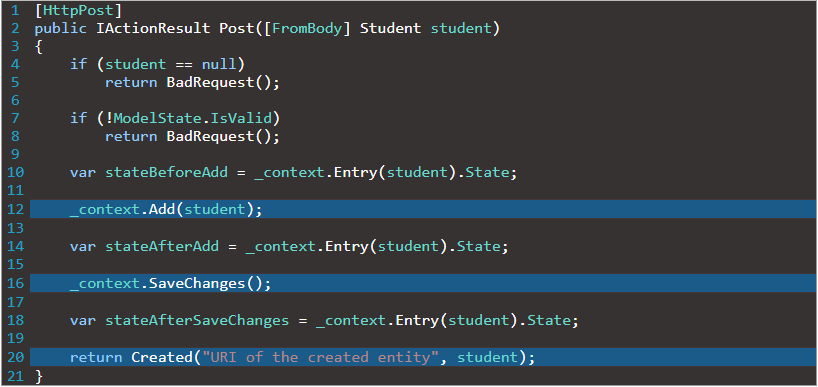
## Create Rows – Modifying Data with Add and AddRange methods

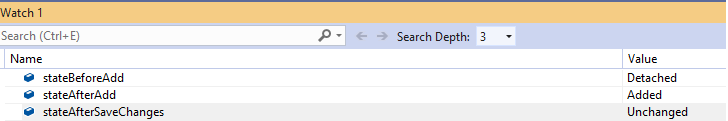
Chúng ta có thể tạo mới các rows trong database băng việc sử dụng Add và AddRangr methods. Add method sẽ chỉnh sửa State



Về cơ bản thằng EF Core thì Context có hàm Add kia nhưng EF thường hình như đếch có

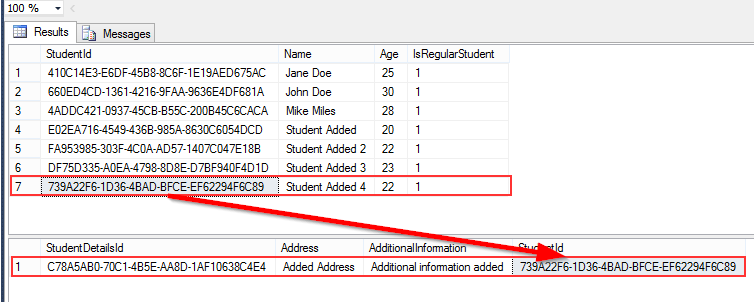
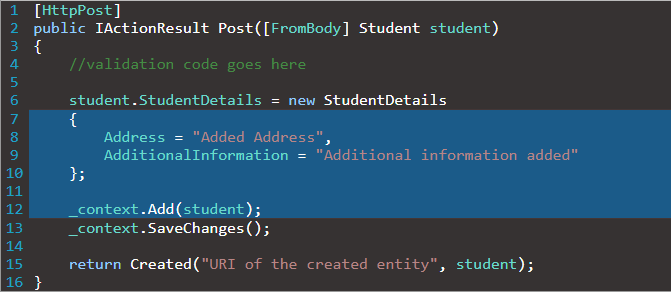
## Tracking Changes when Adding an Entity





Với đoạn mã trên ta kiểm tra việc trước và sau khi SaveChanges thì tracking nó đã thay đổi là từ Detached khi chứa Add, Sau khi Add sẽ là Added, và sau khi SaveChanges nó chuyển thành Unchanged

## Adding Related Entities to the Database



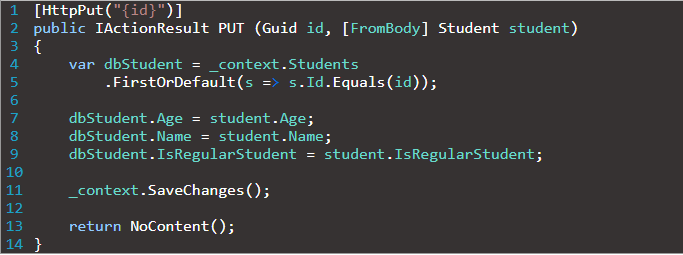
Như chúng ta có thể nhìn bên trên, rõ ràng là chúng ta chỉ adding student object tới context nhưng studentDetails cũng được thêm vào database.

## Updating Rows in the Database

Có vài cách để update rows trong database. Với Connected Update và Disconnected Update. Sự khác nhau giữa chúng là với connected update chúng ta sử dụng the same context object để load và chỉnh sửa entity. Disconnected update thì không trong trường hợp này. Hoặc chúng ta sử dụng các context objects khác nhau hoặc chúng ta nhận một object từ một client cái mà có tất cả properties như là the entity trong database.

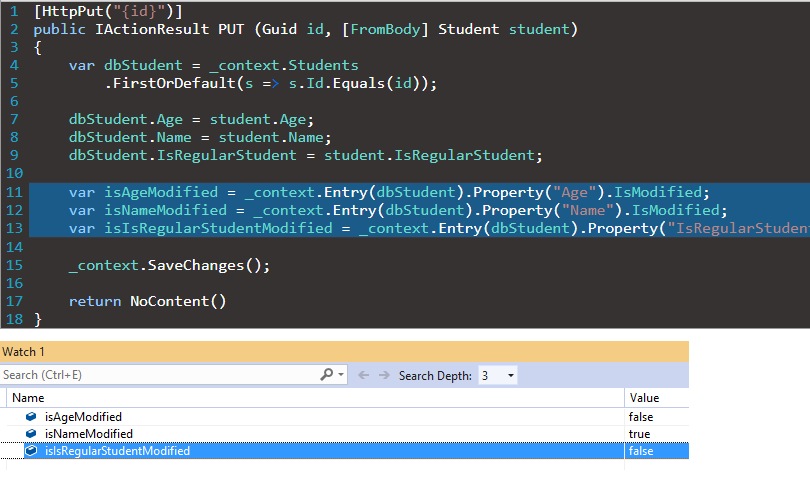
Khi làm việc với connected uodate có 3 bước để execute:

* Read data from database
* Change one or more properties
* Call the SaveChanges method



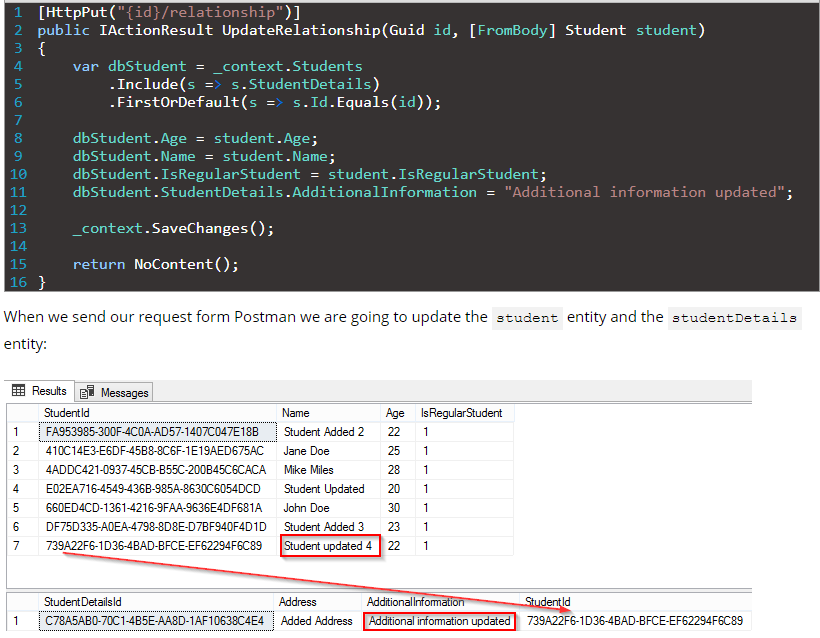
### IsModified Property

Khi chúng ta có một entity cái mà đã Modified state, EFCore sử dụng property khác để cung cấp thông tin về sự thay đổi thực sự. Property này là IsModified property

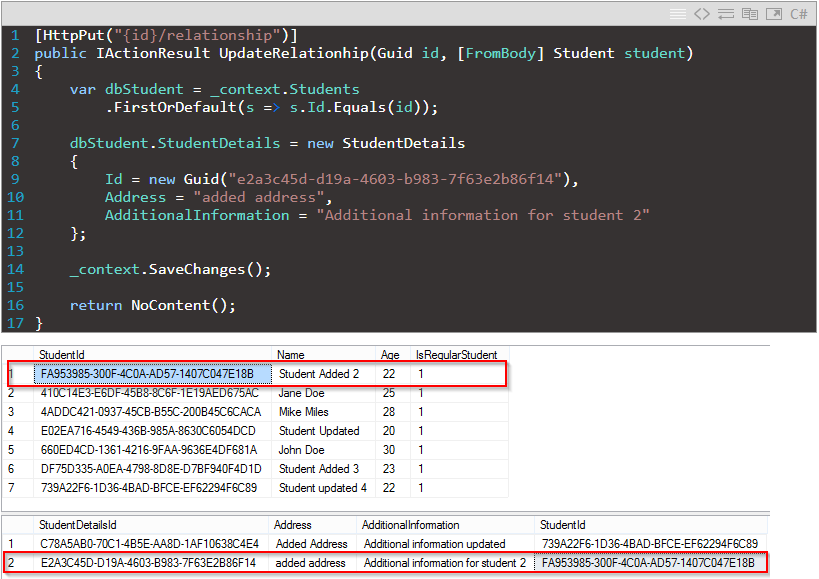


### Updating Relationships in EF Core

Việc Adding realationships tới update operation thì khác dễ dàng. Chúng ta có thể attach a relational entity tới main entity. Modify nó và EF Core sẽ làm tất cả cho chúng ta ngay khi chúng ta call SaveChanges method

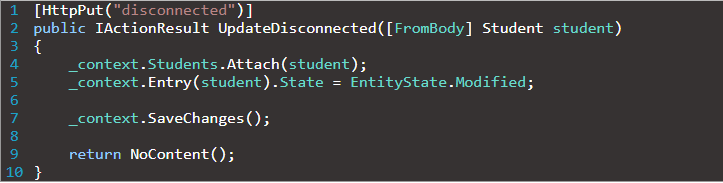


Điều quan trọng cần biết là khi chúng ta update main entity bởi việc adding một new ralational entity, EF Core sẽ tạo một new rơ trong relationship table và connects nó tới main entity vơi foreign key



## Disconnected Update in EF Core

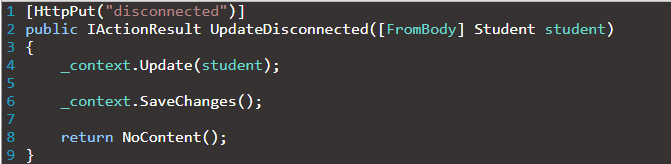
Có một vài cách để execute một disconnected uodate, và chúng ta sẽ show 2 cách để làm điều này. Với ví dụ đầu tiên, chúng ta sẽ attach object được gửi từ clinet, modify state của nó sau đó save nó tới database



Studetn object được gửi từ client thì bị Detached state. Sau đó chúng ta sử dụng Attach method object sẽ chuyển state tới Unchanged.

Điều này cũng có nghĩa rằng tại thời điểm này EF Core bắt đầu tracking the entity. Bây giờ chúng ta sẽ thay đổi state để Modified và save nó tới database.

Một cách khác đề làm điều này là chúng ta sử dụng Update hoặc UpdateRange method



## Delete Operations in EF Core

Chúng ta có 2 cách để delete rows

### Soft Delete

Với cách này, entity bị hidden xa hơn là deleted. Đây không phải là một cách bất thường của việc delete vì thực tế chúng ta không muốn delete một cách thực sử bất cứ một row bởi vì chúng sẽ được yêu cầu cho nhiều thống kê sau này.

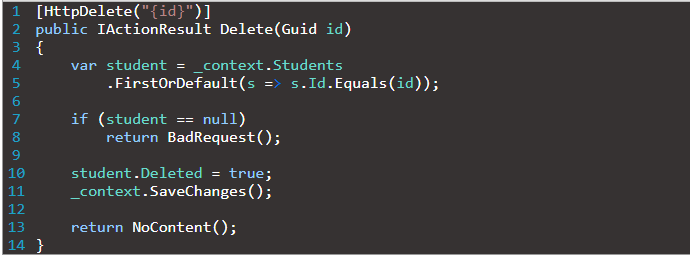
Về cơ bản chúng ta không deleting một entity chúng ta đang update bởi việc chỉnh sửa property của nó



Sau đó chúng ta modify StudentConfiguration class, bằng việc adding một method để filter không lấy thằng có property Deleted là true



Và cuối cùng nếu muốn xóa thì có nghĩa ta update một object là true với Deleted

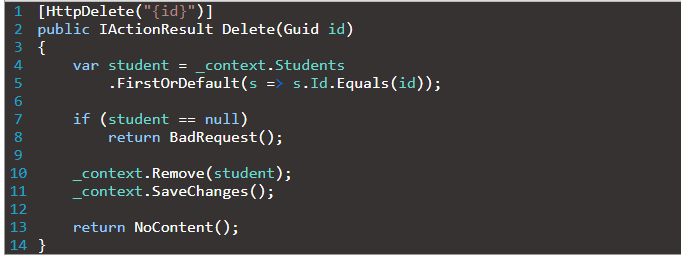


Nếu chúng ta vẫn muốn query thằng kia sau khi đã set filter trong config thì ta có thể làm như sau



### Delete a Single Entity

Với cái này chúng ta thực sự xóa sổ nó khỏi database với Remove và RemoveRange method



Chúng ta phải cẩn thận với Delete actions nếu entity của nó có relationships và chúng ta chưa xác định được họ trong delete action. Việc Delete main entity có thể delete relationships (cascade delete) phục thuộc việc config entity.

### Delete Entity with Relationships

