Th1 code first :  Bạn **viết code C# trước** (class model).

 EF Core sẽ dựa trên class đó để **tạo ra database** (tự động sinh bảng, cột, quan hệ)

* DbContext trong Entity Framework Core là gì

A screenshot of a computer

AI-generated content may be incorrect.

A screen shot of a computer

AI-generated content may be incorrect.

**public DbSet<Test> check { get; set; } // check đại diện Table database trong DbContext**

Trong **Entity Framework (EF Core)**, khi bạn tạo một **class** để mapping với bảng trong database, thì EF **bắt buộc phải biết cột nào là khóa chính (Primary Key)**. Đây là **quy ước (convention)** mặc định của EF.

**📌 Quy ước mặc định (Convention) của EF về Primary Key**

* Nếu class có một thuộc tính tên là:
  + Id
  + ClassNameId (ví dụ: SinhVienId nếu class tên SinhVien)

Thì EF sẽ tự động hiểu đó là **Primary Key** mà không cần bạn phải chỉ định thêm.

👉 Ví dụ:

public class SinhVien

{

public int SinhVienId { get; set; } // EF hiểu đây là PK

public string Ten { get; set; }

public int Tuoi { get; set; }

}

Hoặc:

public class SinhVien

{

public int Id { get; set; } // EF cũng hiểu đây là PK

public string Ten { get; set; }

}

\* public DbSet<Item> Items { get; set; } *// 📌 Tên DbSet: "Items" (số nhiều)*

Connection With Database

ở appsetting.json

"ConnectionStrings": {

"DefaultConnectionString": "Data Source=.;Initial Catalog=myweb\_database;Integrated Security=True;Pooling=False;Encrypt=False;Trust Server Certificate=True"

}

Nếu **bật container** → "ConnectionStrings": {

"DefaultConnectionString": "Server=host.docker.internal;Database=MyApp\_database;Trusted\_Connection=True;TrustServerCertificate=True;"

}

ở program.Class

builder.Services.AddDbContext<MyDbContext>(options => options.UseSqlServer(builder.Configuration.GetConnectionString("DefaultConnectionString") ) );

A screenshot of a computer

AI-generated content may be incorrect.

 Sửa model (thêm cột, sửa bảng).

 Add-Migration "Tên" → lưu thay đổi giống commit.

 Update-Database → áp dụng commit đó vào database.

Drop-Database => xóa toàn bộ database khi bị lỗi

Update-Database

Th2 Databasefirst

Scaffold-DbContext "Data Source=.;Initial Catalog=SchoolDb;Integrated Security=True;TrustServerCertificate=True" Microsoft.EntityFrameworkCore.SqlServer -OutputDir Models -ContextDir Data -DataAnnotations

Nếu database update mà visual cũng muốn cập nhật cái update thì dùng lệnh

Scaffold-DbContext "Data Source=.;Initial Catalog=SchoolDb;Integrated Security=True;TrustServerCertificate=True" Microsoft.EntityFrameworkCore.SqlServer -OutputDir Models -ContextDir Data -DataAnnotations -Force

**Dùng Class DbContext**

**1. Khai báo Controller**

public class ItemsController : Controller

public: Có thể truy cập từ anywhere

class ItemsController: Định nghĩa class controller cho "Items"

: Controller: Kế thừa từ class Controller base của ASP.NET Core MVC

**2. Dependency Injection - Tiêm phụ thuộc**

private readonly MyAppContext \_context;

private: Chỉ accessible trong class này

readonly: Chỉ được gán giá trị một lần (trong constructor)

MyAppContext \_context: Biến để tương tác với database

**3. Constructor - Hàm khởi tạo**

public ItemsController(MyAppContext context)

{

\_context = context;

}

ASP.NET Core tự động inject MyAppContext vào constructor

Dependency Injection pattern: Không tạo context manually mà nhận từ system

Ích lợi: Dễ testing, dễ quản lý vòng đời object

4. Action Method - Index

public IActionResult Index()

{

// Lấy danh sách items từ database rồi chuyển nó thành Collection List

var items = \_context.Items.ToList();

return View(items);

}

**Database Table "Items":**

Id Name Price Category

1 Laptop 1000 Electronics

2 Book 20 Education

3 Coffee 5 Food

Sau khi chạy \_context.Items.ToList():

List<Item> items = new List<Item>

{

new Item { Id = 1, Name = "Laptop", Price = 1000, Category = "Electronics" },

new Item { Id = 2, Name = "Book", Price = 20, Category = "Education" },

new Item { Id = 3, Name = "Coffee", Price = 5, Category = "Food" }

};

**🔧 CÁCH THAY THẾ ĐÚNG:**

**Bước 1: Thêm async và Task**

*// Từ:*

public IActionResult Overview()

*// Thành:*

public async Task<IActionResult> Overview()

**Bước 2: Thêm await trước async methods**

*// Từ:*

var items = \_context.Items.ToList();

*// Thành:*

var items = await \_context.Items.ToListAsync();

**Bước 3: Đổi sang async versions**

*// Sync → Async*

.ToList() → .ToListAsync()

.SaveChanges() → .SaveChangesAsync()

.Find() → .FindAsync()

.FirstOrDefault() → .FirstOrDefaultAsync() : Trả về **phần tử đầu tiên** nếu có.

 Nếu **không có phần tử nào** thì trả về **giá trị mặc định** (default).

* Với **kiểu tham chiếu (class, object)** → null.
* Với **kiểu số (int, double, …)** → 0.
* Với **bool** → false.
*  First() → Lấy phần tử đầu tiên, **nếu không có phần tử sẽ báo lỗi**.
*  FirstOrDefault() → Lấy phần tử đầu tiên, **nếu không có thì trả về default** (an toàn hơn)

 **Int Identity** → EF core tự động tăng khóa chính +=1 mỗi khi insert vào Db

 **String nvarchar** → phải tự gán mỗi khi Insert vào Db

----------------------------------------------------------------------------------------------

Cách các bảng liên kết với nhau trong Visual studio khi sử dụng CRUD

**📊 Database Structure Trực quan**

**Bảng Item (Bảng chính)**

| ItemId (PK) | Name | Price | Description |
| --- | --- | --- | --- |
| 1 | Laptop | 1000 | Gaming laptop |
| 2 | iPhone | 800 | iPhone 15 |
| 3 | Keyboard | 50 | Mechanical |

**Bảng Infor (Bảng phụ - có khóa ngoại)**

| proId (PK) | ItemId (FK) | Thông tin khác... |
| --- | --- | --- |
| 101 | 1 | ... |
| 102 | 1 | ... |
| 103 | 2 | ... |
| 104 | 3 | ... |

**🔄 Flow hoạt động khi thêm mới**

**Bước 1: Thêm Item trước**

csharp

var newItem = new Item

{

Name = "Monitor",

Price = 300,

Description = "27 inch 4K"

};

\_context.Items.Add(newItem);

\_context.SaveChanges(); *// ⚡ Database tự tạo ItemId = 4*

**Bước 2: Database sau khi thêm Item**

| ItemId (PK) | Name | Price | Description |
| --- | --- | --- | --- |
| 1 | Laptop | 1000 | Gaming laptop |
| 2 | iPhone | 800 | iPhone 15 |
| 3 | Keyboard | 50 | Mechanical |
| ✅ **4** | **Monitor** | **300** | **27 inch 4K** |

**Bước 3: Thêm Infor với ItemId vừa tạo**

csharp

var newInfor = new Infor

{

ItemId = newItem.ItemId, *// = 4 (tự động lấy từ Item vừa tạo)*

*// các thuộc tính khác của Infor*

};

\_context.Infors.Add(newInfor);

\_context.SaveChanges();

**Bước 4: Database sau khi thêm Infor**

| proId (PK) | ItemId (FK) | Thông tin khác... |
| --- | --- | --- |
| 101 | 1 | ... |
| 102 | 1 | ... |
| 103 | 2 | ... |
| 104 | 3 | ... |
| ✅ **105** | ✅ **4** | ... |

**🎯 Visual Diagram**

text

[Form Create Item]

↓

[User nhập: Name="Monitor", Price=300]

↓

[Submit] → [Server nhận data không có ItemId]

↓

[Database INSERT] → [Tự sinh ItemId = 4]

↓

[Server lấy được ItemId = 4]

↓

[Thêm Infor với ItemId = 4]

↓

[Database INSERT thành công]

↓

[✅ Hai bảng được kết nối hoàn hảo]

var newItem = new Item { Name = "Laptop", Price = 1000 };

*// Bước 1: Tracking*

\_context.Items.Add(newItem);

*// 📍 Database chưa có gì, chỉ nhớ trong memory*

*// Bước 2: Thực thi*

await \_context.SaveChangesAsync();

*// ⚡ Lúc này mới: INSERT INTO Items (Name, Price) VALUES ('Laptop', 1000)*

*// ⚡ Database thực sự có row mớ*

**Những câu lệnh Add (Insert) , Update, Remove (Delete) , Read trong DbContext**

**1️⃣ \_context.Items.Add(item) Create // không cần có khóa chính ,Db tự tăng +=1**

|  |
| --- |
|  |

|  |
| --- |
|  |

. Items là **DbSet<Item>** → EF biết class Item ánh xạ **bảng Items**.

* Khi gọi Add(item), EF đánh dấu entity là **Added** → SaveChangesAsync() sẽ sinh câu SQL:

INSERT INTO Items (Name, Price) VALUES (@p0, @p1)

* Ở đây **DbSet quyết định bảng nào**.

**2️⃣ \_context.Update(item) Update // Phải có khóa chính, nếu không → EF không biết update → Insert**

**Note : Update tự động hàng trong Table dựa trên điều kiện khóa chính (sql where = khóa chính )**

1. \_context.Items. Update(item)
2. Khi SaveChangesAsync() gọi → EF dùng **metadata của Item** → sinh câu
3. SQL UPDATE:

UPDATE Items

SET Name = @p0, Price = @p1

WHERE ItemId = @p2

**3 \_context.Items.Remove(item) Delete // Phải có khóa chính**

**Remove hàng dựa trên khóa chính**

**SQL : DELETE FROM Items**

**WHERE Id = @id;**

**4 .1 \_ context.Items Read 1 Table // phải có khóa chính**

**SQL**

**SELECT \* FROM Items**

**Where column**

**4.2 \_context.Items .Include (s = > s.nameTable ) // Read 2 Table join với nhau**

**SQL**

**SELECT i.Name , s.Name**

**FROM Item i**

**LEFT JOIN SerialNumber s ON i.SerialNumberId = s.serialNumberId;**

**CRUD**

| **Operation** | **EF Method** | **SQL** | **Mô tả** |
| --- | --- | --- | --- |
| **CREATE** | **Add()** | **INSERT** | **Thêm bản ghi mới** |
| **READ** | **DbSet** | **SELECT** | **Truy vấn dữ liệu** |
| **UPDATE** | **Update()** | **UPDATE** | **Cập nhật bản ghi** |
| **DELETE** | **Remove()** | **DELETE** | **Xóa bản ghi** |

=> Add không cần khóa chính , Update , remove, Read cần khóa chính vì có điều kiện WHERE

Có thể dùng cái này

A screenshot of a computer

AI-generated content may be incorrect.

{

"sdk": {

"version": "8.0.414"

}

}

Cách chỉ Project chạy đúng SDK

Cách xóa Database khi dùng code first

USE master;

GO

ALTER DATABASE databaseName SET SINGLE\_USER WITH ROLLBACK IMMEDIATE;

DROP DATABASE databaseName;