

### 1. EF Core là gì?

EF Core (Entity Framework Core) là một Object-Relational Mapping (ORM) framework mã nguồn mở được phát triển bởi Microsoft, cho phép các nhà phát triển .NET làm việc với cơ sở dữ liệu bằng cách sử dụng các đối tượng .NET. EF Core hỗ trợ nhiều loại cơ sở dữ liệu như SQL Server, SQLite, PostgreSQL, MySQL,...

### 2. Ưu điểm của EF Core so với ADO.NET truyền thống là gì?

* Trừu tượng hóa việc truy vấn, thêm, xóa, cập nhật dữ liệu mà không cần viết SQL trực tiếp.
* Hỗ trợ LINQ để truy vấn dữ liệu.
* Dễ dàng quản lý model và tự động tạo/migration database.
* Hỗ trợ nhiều cơ sở dữ liệu khác nhau.
* Code dễ đọc, bảo trì, và kiểm thử hơn.

### 3. So sánh 2 mô hình: Code First và Database First

|  |  |  |
| --- | --- | --- |
| Tiêu chí | Code First | Database First |
| Cách tiếp cận | Viết Class trước, sinh database sau | Có Database sẵn, sinh ra từ DB |
| Linh hoạt | Cao, dễ thay đổi theo ứng dụng | Phụ thuộc cấu trúc Database ban đầu |
| Công cụ hỗ trợ | Migration command | Scaffold-DbContext |

### 4. Trình bày quy trình triển khai EF Core theo mô hình Code First

1. Cài đặt gói NuGet: Microsoft.EntityFrameworkCore, Microsoft.EntityFrameworkCore.SqlServer, ...
2. Tạo các lớp model.
3. Tạo lớp DbContext.
4. Cấu hình connection string trong appsettings.json.
5. Sử dụng AddDbContext trong Startup.cs.
6. Chạy Add-Migration và Update-Database để tạo database.

### 5. Cách cấu hình chuỗi kết nối trong ASP.NET Core để dùng EF Core?

Trong appsettings.json:

"ConnectionStrings": {

"MyDb": "Server=.;Database=MyDb;Trusted\_Connection=True;"

}

Trong Startup.cs:

services.AddDbContext<MyDbContext>(options =>

options.UseSqlServer(Configuration.GetConnectionString("MyDb")));

### 6. Làm sao để tạo database từ các class model?

* Dùng lệnh:

Add-Migration InitialCreate

Update-Database

* EF Core sẽ tạo database và bảng tương ứng với class model.

### 7. Lệnh nào được dùng? Cách định nghĩa mối quan hệ 1-nhiều và nhiều-nhiều trong Code First?

* Một-nhiều:

public class Author {

public int Id { get; set; }

public List<Book> Books { get; set; }

}

public class Book {

public int Id { get; set; }

public int AuthorId { get; set; }

public Author Author { get; set; }

}

* Nhiều-nhiều (EF Core 5 trở lên):

public class Student {

public int Id { get; set; }

public List<Course> Courses { get; set; }

}

public class Course {

public int Id { get; set; }

public List<Student> Students { get; set; }

}

### 8. Làm sao cập nhật thay đổi model vào database khi dùng Code First?

1. Thay đổi class model.
2. Chạy:

Add-Migration UpdateModel

Update-Database

### 9. DbSet là gì? DbContext là gì?

* **DbSet**: Đại diện cho một bảng trong database.

public DbSet<Student> Students { get; set; }

* **DbContext**: Quản lý kết nối và truy vấn dữ liệu giữa ứng dụng và cơ sở dữ liệu.

public class MyDbContext : DbContext

{

public DbSet<Student> Students { get; set; }

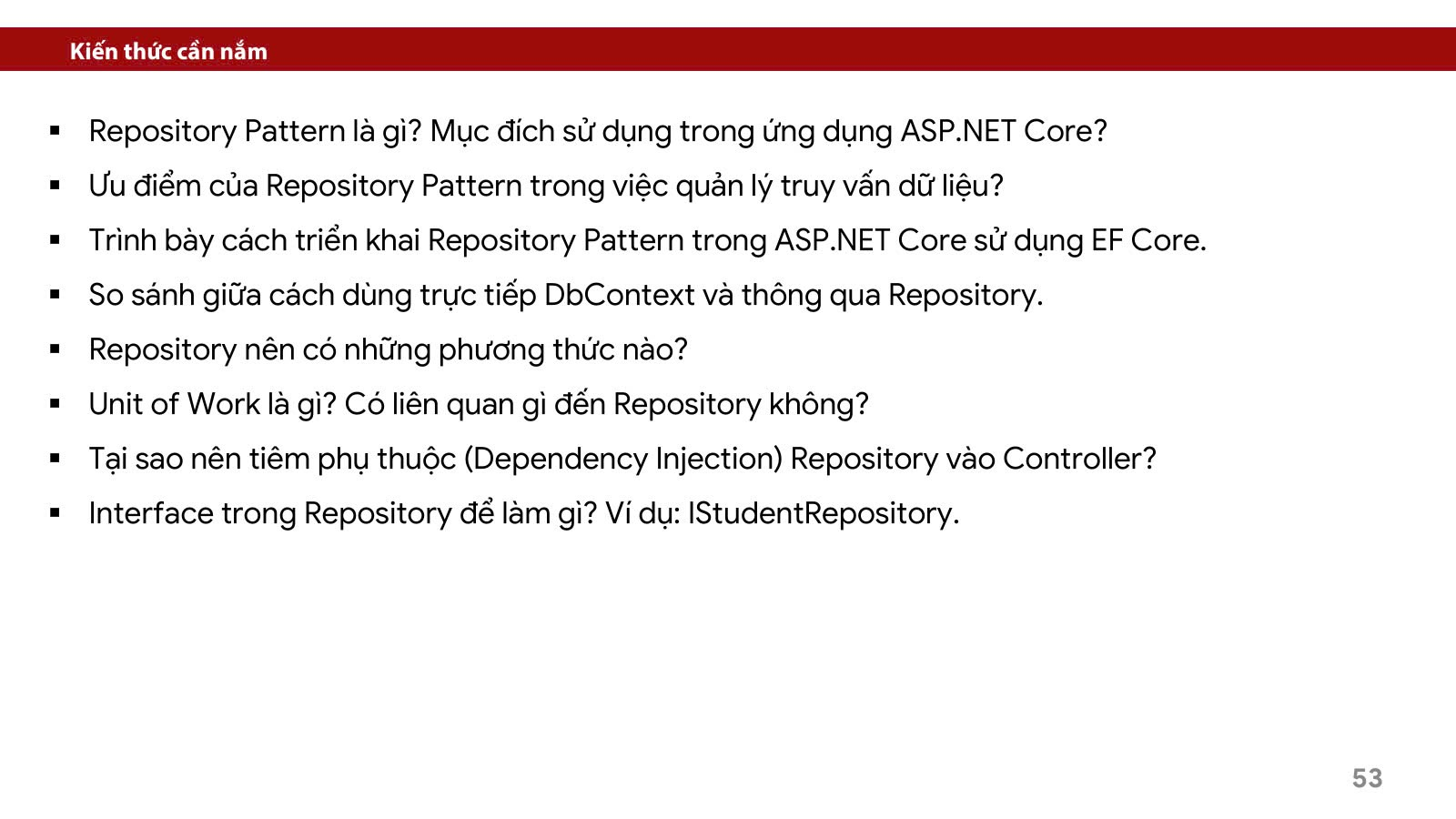
}

### 10. EF Core hỗ trợ những phương thức nào để truy vấn dữ liệu?

* LINQ methods: Where(), First(), FirstOrDefault(), Single(), SingleOrDefault(), ToList(), Any(), Count(), Include() (để load navigation properties),...
* Có thể kết hợp với async: FirstAsync(), ToListAsync(),...

### 11. So sánh First(), FirstOrDefault(), Single(), SingleOrDefault() trong EF Core

| **Phương thức** | **Mô tả** |
| --- | --- |
| First() | Trả về phần tử đầu tiên khớp, nếu không có thì ném lỗi |
| FirstOrDefault() | Trả về phần tử đầu tiên khớp, nếu không có thì trả về null |
| Single() | Trả về đúng 1 phần tử khớp, nếu không có hoặc >1 thì ném lỗi |
| SingleOrDefault() | Trả về đúng 1 phần tử khớp, nếu không có thì null, nếu >1 thì ném lỗi |



## 1. ****Repository Pattern là gì? Mục đích sử dụng trong ASP.NET Core****

* Repository Pattern là một mẫu thiết kế (design pattern) giúp tách **logic truy cập dữ liệu** khỏi **logic xử lý nghiệp vụ**.
* Mục đích:
* Giảm phụ thuộc vào Entity Framework / ORM cụ thể.
* Dễ bảo trì, dễ test.
* Đảm bảo **Separation of Concerns** (tách biệt các tầng).

## 2. ****Ưu điểm của Repository Pattern trong việc quản lý truy vấn dữ liệu****

* Dễ dàng thay đổi nguồn dữ liệu (SQL → NoSQL).
* Tăng tính tái sử dụng và kiểm thử (testability).
* Tập trung logic truy xuất dữ liệu ở một nơi → dễ kiểm soát.

## 3. ****Triển khai Repository Pattern trong ASP.NET Core sử dụng EF Core****

Gồm các bước:

1. Tạo interface: IStudentRepository
2. Tạo lớp triển khai: StudentRepository implements IStudentRepository
3. Trong Startup.cs hoặc Program.cs: đăng ký Dependency Injection

services.AddScoped<IStudentRepository, StudentRepository>();

1. Inject vào Controller:

public class StudentController : Controller

{

private readonly IStudentRepository \_repo;

public StudentController(IStudentRepository repo)

{

\_repo = repo;

}

}

## ****4, So sánh DbContext trực tiếp vs thông qua Repository****

| **Tiêu chí** | **Dùng DbContext trực tiếp** | **Dùng qua Repository** |
| --- | --- | --- |
| Tính gọn nhẹ | Có | Ít hơn |
| Tái sử dụng | Khó | Dễ |
| Kiểm thử (unit test) | Khó | Dễ |
| Tách biệt tầng | Không rõ ràng | Rõ ràng |

## 5. ****Repository nên có những phương thức nào?****

Tuỳ vào domain, nhưng thường gồm:

* GetAll()
* GetById(id)
* Add(entity)
* Update(entity)
* Delete(id)
* Save() (nếu không dùng Unit of Work)

## ****6.Unit of Work là gì? Có liên quan đến Repository không?****

* **Unit of Work** là một pattern dùng để **quản lý các thao tác giao dịch (transaction)** như một đơn vị công việc duy nhất.
* Khi dùng nhiều Repository, Unit of Work đảm bảo rằng:

\*Tất cả thay đổi sẽ được commit cùng lúc (hoặc rollback nếu lỗi).

* Thường được implement với 1 SaveChanges() duy nhất.

## 7. ****Tại sao nên tiêm phụ thuộc Repository vào Controller?****

* Giúp controller không phụ thuộc trực tiếp vào lớp cụ thể (như DbContext).
* Dễ dàng thay đổi implement (mock khi test).
* Tuân theo nguyên lý **Dependency Inversion** trong SOLID.

## ****8 Interface trong Repository để làm gì?****

## ****(Ví dụ: IStudentRepository)****

* Định nghĩa hành vi mà Repository cần triển khai.
* Giúp tách **contract** khỏi **implementation**.
* Hỗ trợ Dependency Injection.
* Dễ dàng mock khi test unit:

public interface IStudentRepository

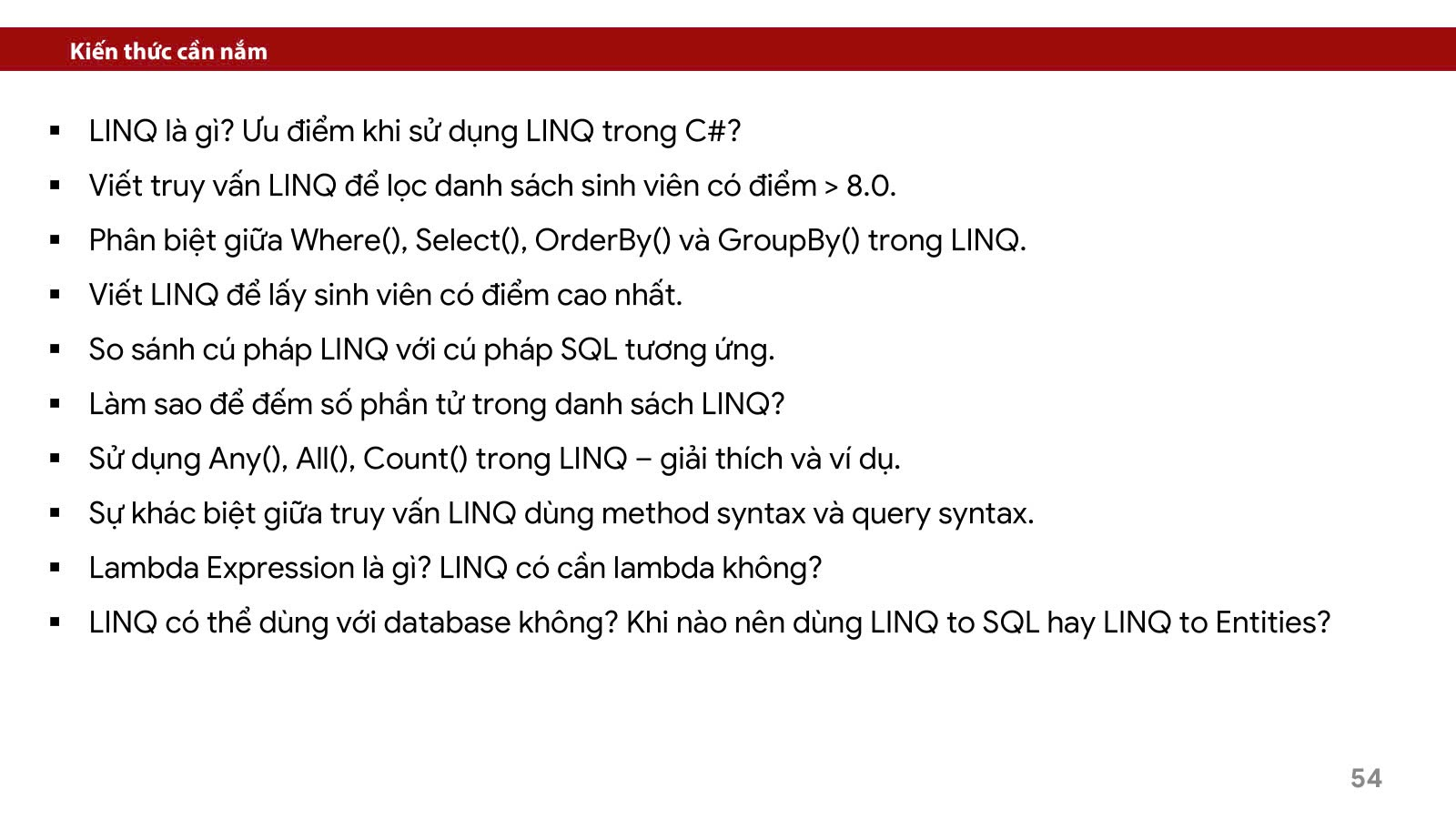
{

IEnumerable<Student> GetAll();

Student GetById(int id);

void Add(Student student);

}



## 1. LINQ là gì? Ưu điểm khi dùng LINQ trong C#

* **LINQ (Language Integrated Query)** là cú pháp truy vấn dữ liệu tích hợp trong C#.
* Ưu điểm:
* Viết truy vấn ngay trong C# (không cần SQL).
* Tự động kiểm tra lỗi khi biên dịch.
* Làm việc với nhiều nguồn dữ liệu (collections, arrays, DB…).

## 2. Viết truy vấn LINQ lọc sinh viên có điểm > 8.0

var result = students.Where(s => s.Score > 8.0);

## 3. Phân biệt Where(), Select(), OrderBy(), GroupBy()

| **Phương thức** | **Mục đích** |
| --- | --- |
| Where() | Lọc dữ liệu |
| Select() | Chọn dữ liệu cần thiết |
| OrderBy() | Sắp xếp tăng dần |
| GroupBy() | Nhóm dữ liệu theo một khóa |

## 4. LINQ lấy sinh viên có điểm cao nhất

var maxScoreStudent = students.OrderByDescending(s => s.Score).First();

## 5. So sánh cú pháp LINQ vs SQL

**LINQ method syntax**:

students.Where(s => s.Score > 8);

**LINQ query syntax** (gần giống SQL):

from s in studentswhere s.Score > 8select s;

## 6. Làm sao để đếm số phần tử trong danh sách LINQ?

int count = students.Count();

## 7. Sử dụng Any(), All(), Count() – Giải thích

| **Hàm** | **Ý nghĩa** | **Ví dụ** |
| --- | --- | --- |
| Any() | Có ít nhất 1 phần tử thỏa điều kiện | students.Any(s => s.Score > 8) |
| All() | Tất cả thỏa điều kiện | students.All(s => s.Score > 5) |
| Count() | Đếm số phần tử | students.Count() |

## 8. Sự khác biệt: method syntax vs query syntax

* **Method Syntax**: Dạng gọi hàm, phổ biến hơn, linh hoạt.
* **Query Syntax**: Giống SQL, dễ đọc hơn với người quen SQL.

## 9. Lambda Expression là gì? LINQ có cần lambda không?

* Lambda là biểu thức ẩn danh (ví dụ: s => s.Score > 8)
* LINQ method syntax gần như **luôn cần** lambda.
* Query syntax thì **không cần lambda**.

## 10. LINQ có thể dùng với database không?

* Có, với:

**LINQ to SQL**

**LINQ to Entities** (EF Core)

* **Khi nào dùng LINQ to SQL hay LINQ to Entities?**

LINQ to SQL: chỉ hỗ trợ SQL Server, không phát triển nữa.

LINQ to Entities (Entity Framework): mạnh mẽ hơn, đa nền tảng → **nên dùng**.