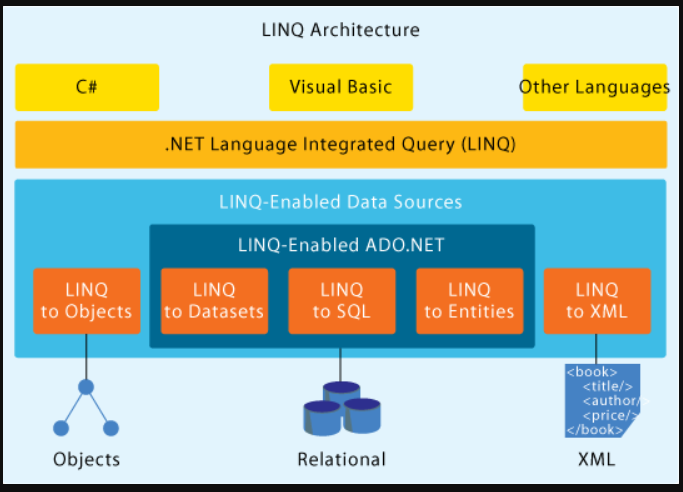
LINQ

1. **LinQ là gì?**

* Là ngôn ngữ mở rộng cho các ngôn ngữ lập trình C# và Visual Basic.Net ,cung cấp khả năng truy vấn trực tiếp dữ liệu Obj, csdl, dataset và XML.
* Giúp tăng tốc thời gian viết mã lệnh(chưa chắc hiệu quả về tốc độ xử lí).

1. **Kiến trúc và các thành phần trong LinQ:**

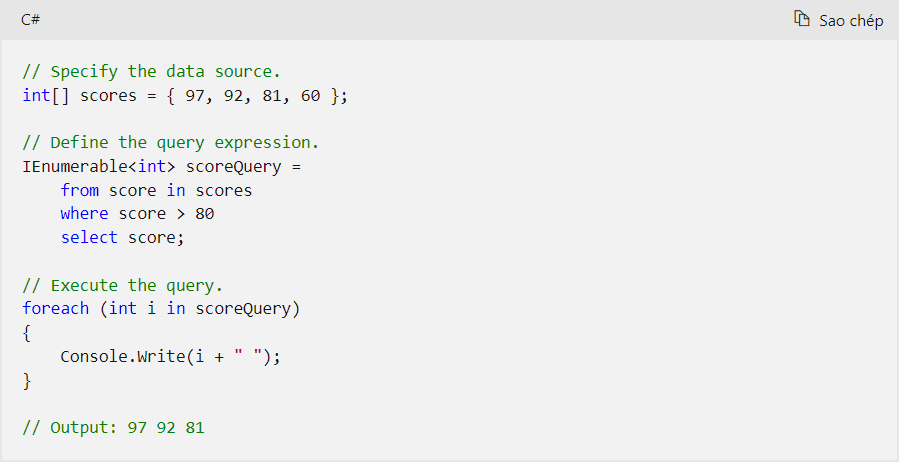


4 tầng:tầng 1:hỗ trợ các ngôn ngữ, tầng 2:Standard Query Operators Expression Tress, tầng 3:Tầng trung gian , tầng 4: CSDL/Nguồn

1. **Các thư viện hỗ trợ trong LinQ:**

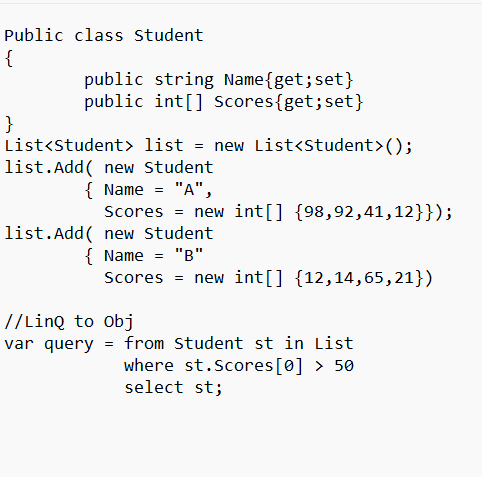
* System.Linq:Hỗ trợ sử dụng các Obj
* System.Data.Linq: Hỗ trợ sử dụng các csdl quan hệ
* System.Data.Obj: Hỗ trợ sd các entities
* System.XML.Linq: Hỗ trợ sd XML

1. **VD:**

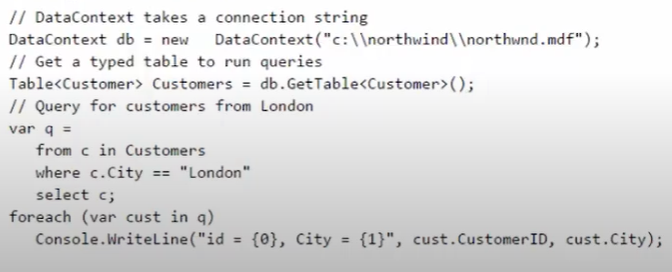


1. **Hệ sinh thái Linq:**

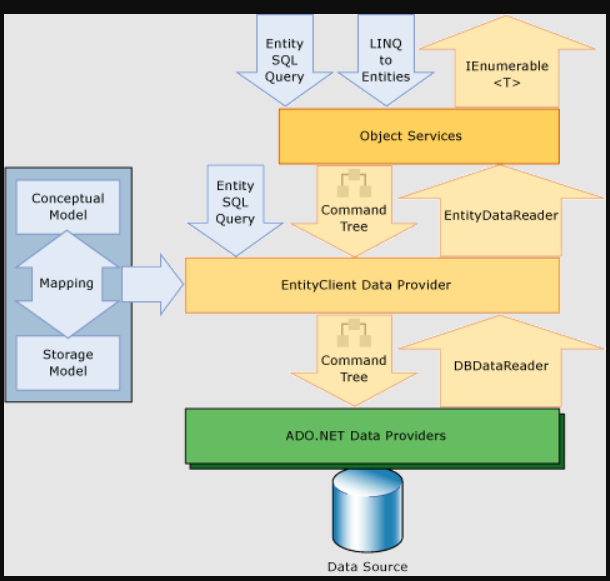
* Linq to Obj: Các cách thức truy vấn các Collection có dạng Ienumerable hoặc Ienumerable<T>, có thể là các Array,List, ArrayList,Dictionary....



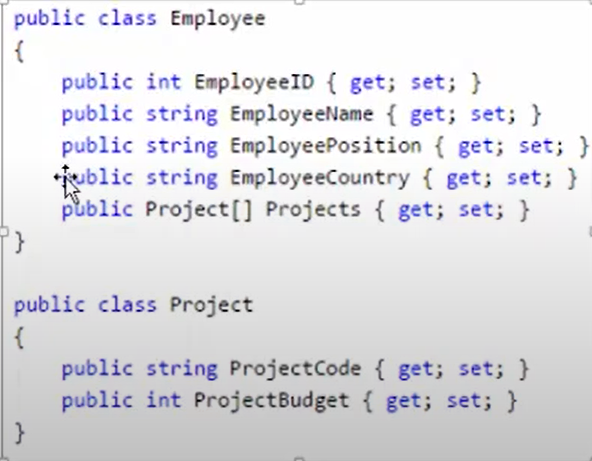
* Linq to SQL: Cho phép mô hình hóa + truy vấn + tương tác vs csdl , hỗ trợ đầy đủ transaction, view và các stored procedure

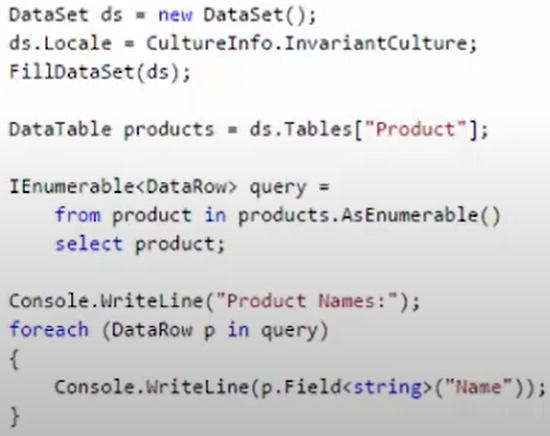


* Linq to Entity: Là 1 nền tảng được sử dụng để làm việc vs database thông qua cơ chế ánh xạ ORM,ta có thể truy vấn thao tác vs database gián tiếp thông qua các đối tượng lập trình .Cho phép ta có thể truy vấn các thực thể bên trong EF



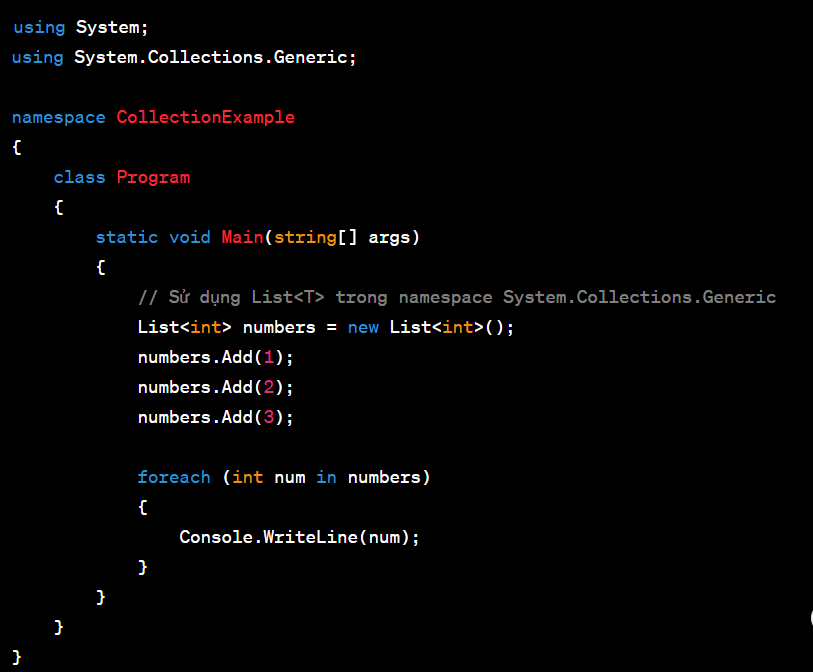
* Linq to XML: Cho phép ghi, đọc XML 1 cách dễ dàng, có khả năng kiểm tra tính hợp lệ XML



* Linq to Dataset: Cho phép sử dung Dataset như 1 nguồn dữ liệu bình thường bằng các cú pháp truy vấn căn bản của Linq
* 

1. **Generics:**

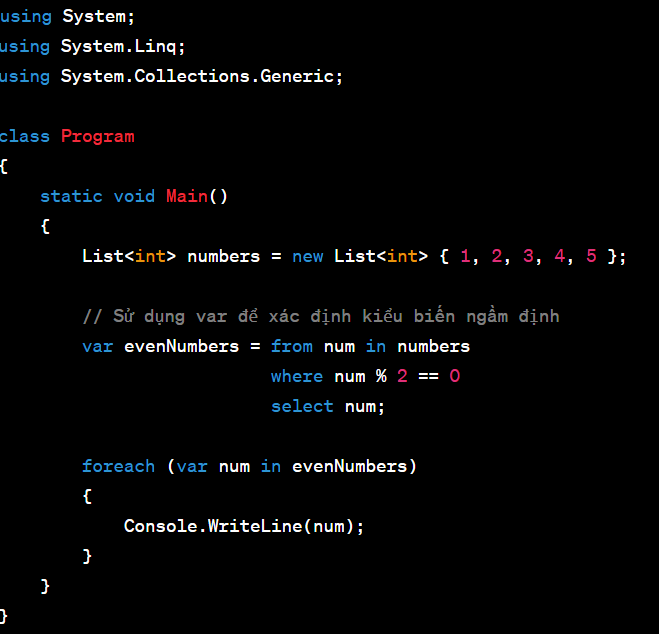
* Generics trong LINQ cho phép bạn viết các truy vấn linh hoạt và tái sử dụng cho các kiểu dữ liệu khác nhau một cách an toàn kiểu. Điều này giúp tạo ra mã linh hoạt và hiệu quả khi thực hiện các hoạt động truy vấn trên các tập hợp và nguồn dữ liệu khác nhau.
* **using System.Collections.Generic:** là một khai báo dùng để import (hoặc sử dụng) namespace System.Collections.Generic trong ngôn ngữ lập trình C#. Namespace này chứa các loại (types) và lớp (classes) liên quan đến các cấu trúc dữ liệu tập hợp được xây dựng trên cơ sở của generics.
* Trong C#, tập hợp các cấu trúc dữ liệu được triển khai trong namespace này bao gồm các loại như List<T>, Dictionary<TKey, TValue>, HashSet<T> và nhiều loại khác. Các cấu trúc này cung cấp các phương thức và thuộc tính để thao tác với dữ liệu tập hợp một cách hiệu quả.



* Trong đoạn mã trên, việc sử dụng **using System.Collections.Generic;** cho phép chúng ta sử dụng **List<int>** mà không cần phải ghi đầy đủ đường dẫn **namespace (System.Collections.Generic.List<int>)**. Thay vào đó, chúng ta có thể viết gọn là **List<int>.**

1. **Implicitly Typed Variables trong LinQ:**

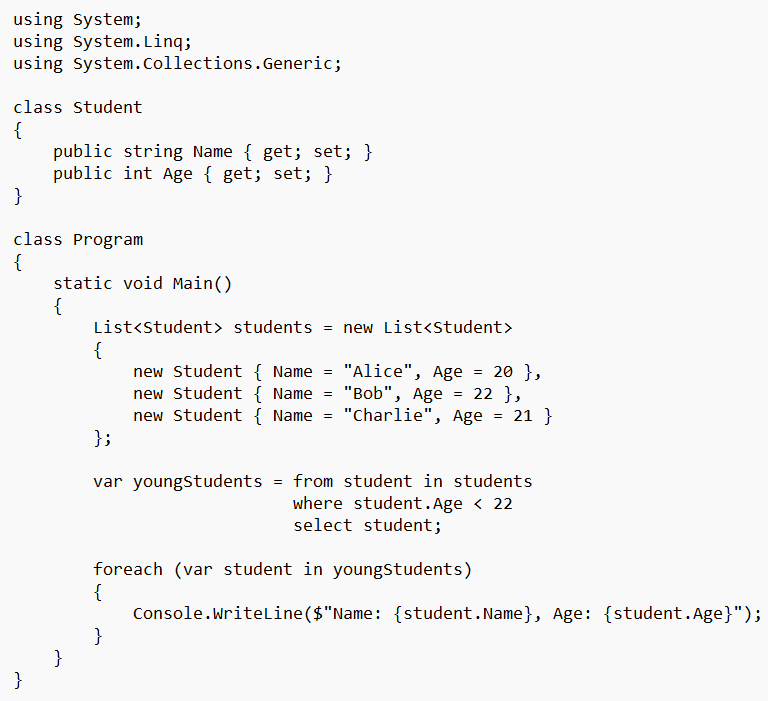
* (Biến được xác định ngầm định) trong LINQ là một tính năng trong ngôn ngữ lập trình C# cho phép bạn khai báo biến mà không cần xác định rõ ràng kiểu dữ liệu của biến đó. Thay vào đó, trình biên dịch sẽ tự động suy ra kiểu dữ liệu dựa trên giá trị bạn gán cho biến.
* Implicitly Typed Variables thường được sử dụng trong LINQ để tiết kiệm thời gian viết mã và làm cho mã dễ đọc hơn trong những tình huống mà bạn biết rõ kiểu dữ liệu của biến.



Ở ví dụ trên, var **evenNumbers** sẽ tự động xác định kiểu của biến dựa trên kết quả của truy vấn LINQ. Trình biên dịch sẽ suy ra rằng **evenNumbers** là một **IEnumerable<int>,** vì vậy bạn có thể sử dụng nó trong một **vòng lặp foreach** để lặp qua các số chẵn.

1. **Object Initializers trong LinQ:**

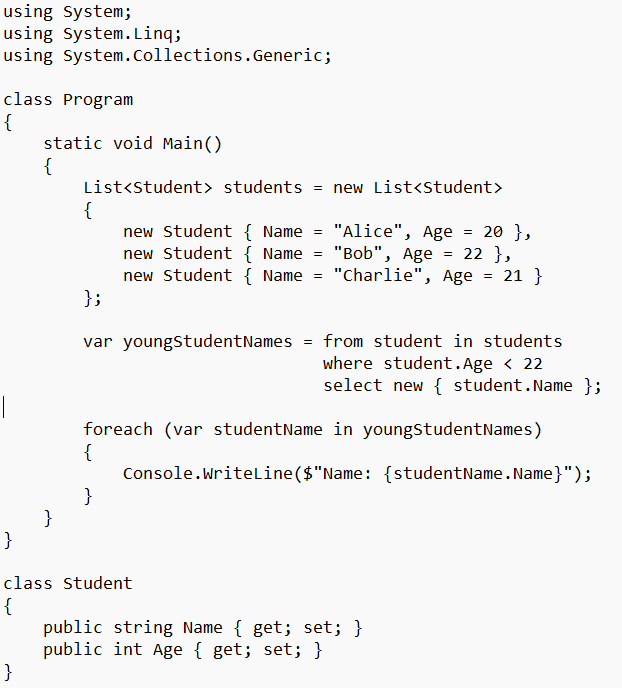
* Là một tính năng trong C# cho phép bạn khởi tạo các thuộc tính của một đối tượng một cách ngắn gọn và dễ đọc. Điều này thường được sử dụng trong LINQ khi bạn tạo đối tượng mới và cần thiết lập các thuộc tính của nó một cách nhanh chóng.



* Trong ví dụ trên, chúng ta sử dụng **Object Initializers** để khởi tạo các thuộc tính của đối tượng **Student** khi thêm các sinh viên vào danh sách **students**. Việc này giúp ta tạo và thiết lập đối tượng một cách rõ ràng và ngắn gọn khi làm việc với LINQ.

1. **Anonymous Types trong LinQ:**

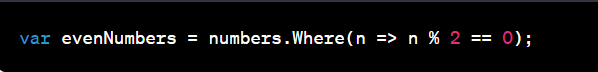
* (kiểu ẩn danh) là một tính năng trong C# cho phép bạn tạo ra các đối tượng mà không cần định nghĩa một kiểu lớp cụ thể trước. Các kiểu ẩn danh thường được sử dụng trong LINQ khi bạn muốn tạo ra các cấu trúc dữ liệu tạm thời để chứa kết quả truy vấn hoặc dữ liệu từ nhiều nguồn.



* Trong ví dụ trên, chúng ta sử dụng **Anonymous Types** để tạo ra các đối tượng không có kiểu lớp cụ thể trong truy vấn LINQ. Việc này cho phép ta chọn và tạo ra chỉ những thông tin mà ta cần, trong trường hợp này là tên của những **student** có tuổi dưới 22. Chúng ta không cần phải định nghĩa một kiểu lớp mới cho mục đích này.

1. **Extension Methods trong LinQ:**

* Một vai trò quan trọng trong việc xử lý và truy vấn dữ liệu. Chúng giúp biến việc truy vấn dữ liệu thành các phương thức tiện ích có thể được gọi trực tiếp trên các tập hợp (collections) và cung cấp cú pháp dễ đọc và mạch lạc hơn.
* Các extension methods trong LINQ thường thao tác với các tập hợp dữ liệu (**IEnumerable<T>** hoặc **IQueryable<T**>) và cho phép bạn thực hiện các thao tác truy vấn như lọc, sắp xếp, nhóm, kết hợp và biến đổi dữ liệu một cách dễ dàng.
* **Where**: Dùng để lọc các phần tử thỏa mãn điều kiện cho trước.



* **OrderBy** / OrderByDescending: Dùng để sắp xếp các phần tử.



* **Select**: Dùng để chọn ra một phần hoặc biến đổi dữ liệu.



* **GroupBy**: Dùng để nhóm các phần tử theo một thuộc tính hoặc biểu thức.



* **Join**: Dùng để kết hợp hai tập hợp dữ liệu dựa trên một hoặc nhiều thuộc tính chung.

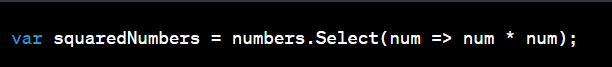
var joinedData = orders.Join(customers, order => order.CustomerId, customer => customer.Id, (order, customer) => new { Order = order, Customer = customer });

1. **Delegate trong linq:**

* Một vai trò quan trọng để xác định các thao tác xử lý dữ liệu hoặc điều kiện mà bạn muốn áp dụng cho các phần tử trong tập hợp. Các delegate thường được sử dụng trong các phương thức như Where, Select, OrderBy, và nhiều phương thức khác để thực hiện các tác vụ truy vấn dữ liệu.
* Predicate Delegate trong **Where**:Delegate trong linq:
* Trong phương thức **Where**, bạn sử dụng một delegate kiểu **Func<T, bool>** để xác định điều kiện mà phần tử cần thỏa mãn để được lựa chọn.



* Func Delegate trong **Select**:
* Trong phương thức **Select**, bạn sử dụng delegate kiểu **Func<T, TResult>** để biểu thị cách biến đổi mỗi phần tử trong tập hợp.



* Comparison Delegate trong **OrderBy**:
* Trong phương thức **OrderBy**, bạn sử dụng delegate kiểu **Comparison<T>** để so sánh hai phần tử.



* Action Delegate trong **ForEach**:
* Trong phương thức **ForEach**, bạn sử dụng delegate kiểu **Action<T>** để biểu thị hành động mà bạn muốn thực hiện trên từng phần tử.



* Delegate trong LINQ giúp bạn có khả năng định nghĩa logic xử lý dữ liệu hoặc điều kiện một cách linh hoạt và trực tiếp trong các phương thức truy vấn. Điều này làm cho mã trở nên dễ đọc và dễ bảo trì, và cho phép bạn tập trung vào việc xử lý dữ liệu mà không cần phải tạo ra các vòng lặp hay điều kiện phức tạp.

1. **Lambda Expressions trong LinQ:**

* Là một phần quan trọng của LINQ (Language Integrated Query). Chúng được sử dụng để tạo các hàm nặc danh (anonymous functions) để biểu thị logic xử lý dữ liệu trong các phương thức LINQ như **Where**, **Select**, **OrderBy**, và nhiều phương thức khác.
* Lambda expressions có cú pháp ngắn gọn và tiện lợi, giúp việc định nghĩa các hàm nặc danh trở nên dễ dàng hơn.
* Dưới đây là cú pháp cơ bản của lambda expressions trong C#:



* **Where**:
* Lambda expression được sử dụng để xác định điều kiện trong phương thức **Where**.



* **Select**:
* Lambda expression được sử dụng để biểu thị cách biến đổi mỗi phần tử trong phương thức **Select**.



* **OrderBy**:
* Lambda expression được sử dụng để so sánh và xác định trật tự sắp xếp trong phương thức **OrderBy**.



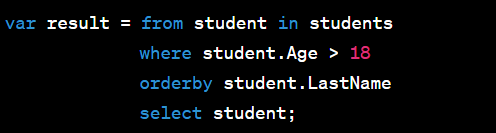
* **GroupBy**:
* Lambda expression được sử dụng để xác định cách nhóm các phần tử trong phương thức **GroupBy**.



* **Lambda expressions** có thể sử dụng cú pháp khá linh hoạt và ngắn gọn, cho phép bạn biểu thị logic xử lý dữ liệu một cách dễ dàng và đọc được trong mã. Chúng thường được sử dụng rộng rãi trong LINQ để tạo ra các hàm nặc danh mà không cần phải định nghĩa các phương thức riêng biệt.

1. **Query syntax trong linq:**

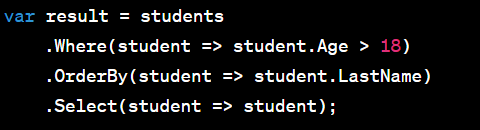
* Query syntax cho phép bạn viết truy vấn dưới dạng cú pháp tương tự như câu truy vấn SQL



* Trong ví dụ trên:
* **from student in students**: Điều này tương đương với việc chọn mỗi phần tử student từ tập hợp students.
* **where student.Age > 18**: Lọc các sinh viên có tuổi lớn hơn 18.
* **orderby student.LastName**: Sắp xếp kết quả theo trường LastName của sinh viên.
* **select student**: Chọn các sinh viên thỏa mãn điều kiện đã xác định.
* Cú pháp query syntax sử dụng từ khóa **from**, **where**, **orderby**, và **select** để biểu thị các phần khác nhau của truy vấn. Cú pháp này rất thuận tiện khi bạn muốn viết các truy vấn có cấu trúc phức tạp, và nó giúp mã trở nên dễ đọc hơn đặc biệt khi truy vấn có nhiều bước khác nhau.

1. **Method syntax trong linq:**

* **method syntax** là cách thứ hai để viết các truy vấn dựa trên các phương thức mở rộng (extension methods). Method syntax sử dụng các phương thức mở rộng như **Where**, **Select**, **OrderBy**, và nhiều phương thức khác để thực hiện các tác vụ truy vấn.
* Dưới đây là một ví dụ về method syntax trong LINQ:



* Trong ví dụ trên:
* .**Where(student => student.Age > 18)**: Lọc các sinh viên có tuổi lớn hơn 18.
* .**OrderBy(student => student.LastName)**: Sắp xếp kết quả theo trường LastName của sinh viên.
* .**Select(student => student):** Chọn các sinh viên thỏa mãn điều kiện đã xác định.
* **Method syntax** sử dụng các phương thức mở rộng như là các phần của truy vấn. Cách này thường trông giống với việc gọi các phương thức liên quan đến tập hợp dữ liệu. Method syntax thích hợp khi bạn muốn viết các truy vấn đơn giản và ngắn gọn.
* Cả **query syntax và method syntax** đều thực hiện cùng một công việc, chỉ có cú pháp khác nhau.

1. **Deferred execution và Lazy loading trong linq:**

* Deferred execution và lazy loading là hai khái niệm quan trọng liên quan đến việc thực hiện truy vấn dữ liệu trong LINQ. Chúng liên quan đến việc khi nào truy vấn thực sự được thực hiện và dữ liệu được tải lên từ nguồn dữ liệu.
* **Deferred Execution (Thực hiện trì hoãn):**
* Deferred execution là tính năng trong LINQ cho phép truy vấn được xây dựng và định nghĩa, nhưng thực tế thực hiện truy vấn và tải dữ liệu từ nguồn không xảy ra cho đến khi dữ liệu thực sự cần được sử dụng.
* Khi bạn xây dựng một truy vấn LINQ, bạn đang chỉ tạo ra một tập hợp các phép biến đổi, điều kiện và xử lý dữ liệu. Truy vấn thực sự không thực hiện cho đến khi bạn yêu cầu dữ liệu bằng cách lặp qua kết quả hoặc gọi các phương thức kết quả như **ToList(), ToArray(), FirstOrDefault(),** vv.
* Deferred execution giúp tối ưu hóa hiệu suất bằng cách trì hoãn việc tải dữ liệu từ nguồn cho đến khi nó thực sự cần thiết.
* **Lazy Loading (Tải lười biếng):**
* Lazy loading là một khái niệm liên quan đến việc tải dữ liệu từ cơ sở dữ liệu hoặc nguồn dữ liệu khác theo nhu cầu thực sự của ứng dụng.
* Trong ngữ cảnh LINQ, khi dữ liệu được yêu cầu, nó được tải lên từ nguồn dữ liệu chứ không phải tải tất cả dữ liệu một lúc. Điều này có lợi cho hiệu suất và tối ưu hóa tài nguyên. Lazy loading thường được thực hiện tự động trong nhiều trường hợp, đặc biệt là khi bạn thao tác với dữ liệu dạng ORM (Object-Relational Mapping).
* Tóm lại, deferred execution và lazy loading là hai khái niệm quan trọng trong LINQ và các hệ thống tương tự. Deferred execution cho phép trì hoãn việc thực hiện truy vấn cho đến khi dữ liệu thực sự cần, trong khi lazy loading tải dữ liệu từ nguồn dữ liệu khi cần thiết để tối ưu hóa hiệu suất và tài nguyên.