

XÂY DỰNG HTTP TRÊN CÁC FRAMEWORK

Kết Nối CSDL

Giảng viên: *Mai Xuân Hùng*
Mail: *hungmx@uit.edu.vn*

Nội dung



ADO.NET



LINQ



Entity Framework



ADO.NET

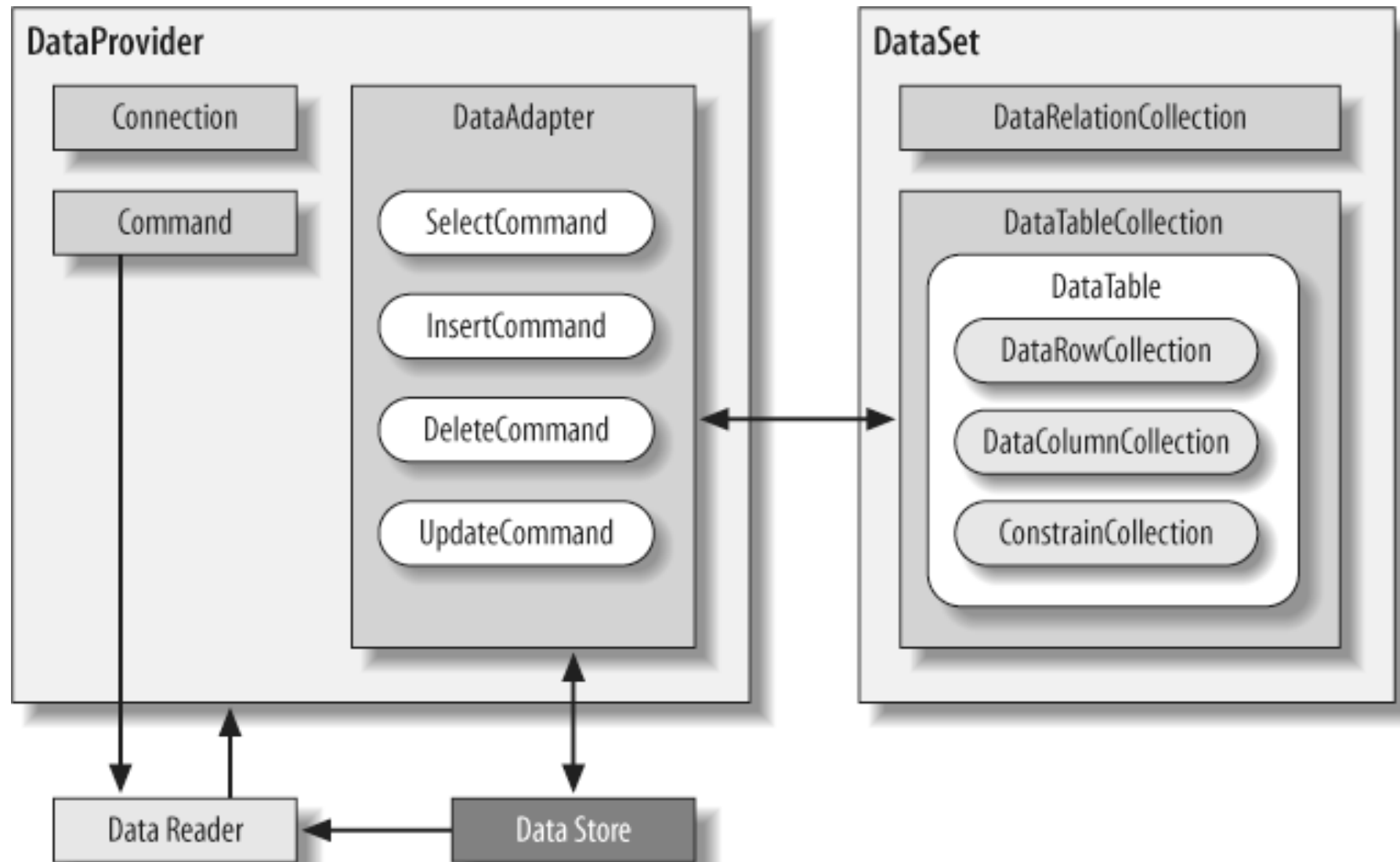
ADO.NET

- ❖ Là thành phần trung gian được dùng để ứng dụng C# thao tác được với nguồn dữ liệu (data source).
- ❖ Có thể tương tác với nhiều nguồn dữ liệu khác nhau: text, excel, XML hoặc một CSDL (database)
- ❖ Thao tác được với các CSDL trên các hệ quản trị CSDL: SQL Server, MySQL, Oracle, SQLite....
- ❖ Ứng với mỗi loại nguồn CSDL sẽ có các Data Provider thích hợp.
- ❖ Tương tác được trên các CSDL theo ODBC hoặc OLE DB.

Kiến trúc ADO.NET

- ❖ Gồm nhiều phần rời rạc nhau, mỗi phần có thể sử dụng độc lập hoặc đồng thời nhiều thành phần được sử dụng.
- ❖ Được chia làm hai thành phần tương ứng cho 2 hình thức kết nối với nguồn dữ liệu:
 - Thành phần kết nối (connected): cho kiến trúc **connected** (Connected Architecture)
 - Thành phần cục bộ (disconnected): cho kiến trúc **disconnected** (Disconnected Architecture)
- ❖ Hai thành phần này tương tác được với nhau thông qua thành phần **DataAdapter**.

Kiến trúc ADO.NET



Connected

- ❖ Là cách thức truy cập trực tiếp vào cơ sở dữ liệu (mở một connection) để truy vấn dữ liệu.
- ❖ Thực hiện trực tiếp các câu lệnh truy vấn dữ liệu (thêm, xóa, sửa và lấy dữ liệu) khi đang kết nối ứng dụng với CSDL.
- ❖ Truy vấn dữ liệu nhanh.
- ❖ Tạo ra nhiều kết nối vào CSDL.



Connected Architecture

Thành phần Connected

❖ Connection:

- Thực hiện kết nối tới CSDL, đóng/mở kết nối, kiểm tra tình trạng kết nối.
- Xác định các thông số kết nối: hệ quản trị, tên CSDL, username, password của CSDL và các tham số cần thiết khác.

❖ Command:

- Chịu trách nhiệm thực thi các truy vấn: thêm, xóa, sửa, lấy dữ liệu.
- Làm việc với cấu trúc của CSDL: thay đổi cấu trúc các bảng.
- Hoạt động trên một connection cụ thể.

❖ Parameter:

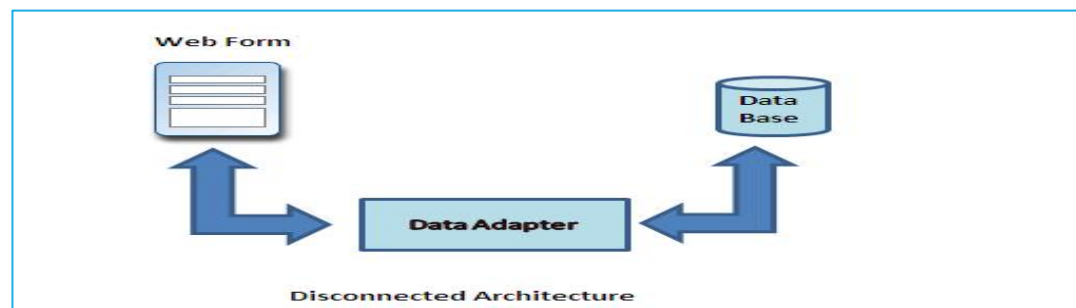
- Truyền tham số cho các truy vấn một cách linh hoạt và an toàn.
- Hoạt động trên các Command, đưa tham số và các Command.

Thành phần Connected

- ☞ Phụ thuộc vào hệ quản trị mà ứng dụng đang kết nối sẽ có các lớp tương ứng :
 - SQL Server sẽ có các **object** được tạo ra từ class **SqlConnection**.
 - MySQL sẽ có các **object** được tạo ra từ class **MySqlConnection**
 - Oracle sẽ có các **object** từ class **OracleConnection**

Disconnected

- ❖ Là cách truy vấn dữ liệu khi ứng dụng không cần phải kết nối trực tiếp với CSDL.
- ❖ Dữ liệu trong CSDL trước khi được truy vấn sẽ được tạo ra một bản sao trong bộ nhớ.
- ❖ Thực hiện các câu lệnh truy vấn dữ liệu (thêm, xóa, sửa và lấy dữ liệu) trên bản sao này.
- ❖ Sau khi kết thúc truy vấn dữ liệu sẽ mở kết nối và cập nhật toàn bộ bản sau này xuống lại CSDL



Thành phần **Disconnected**

- ❖ Để tạo ra **bản sao** của CSDL, kiến trúc **Disconnected** sẽ chứa các **class** mô phỏng cấu trúc của CSDL.
- ❖ **DataSet**: chứa đựng toàn bộ cơ sở dữ liệu.
- ❖ **DataTable**: chứa dữ liệu theo bảng.
- ❖ **DataRow**: lấy dữ liệu và truy vấn theo dòng.
- ❖ **DataColumn**: lấy dữ liệu và truy vấn dữ liệu theo cột.
- ❖ **DataView**: cho phép dữ liệu sau khi được đưa vào DataSet hoặc DataTable có thể xem theo nhiều cách khác nhau; dữ liệu có thể sắp xếp dựa trên giá trị cột hoặc lọc theo tập dữ liệu con theo các tiêu chí được chỉ định.
- ❖ **DataRelation**: chỉ ra mối quan hệ của các DataTable trong dữ liệu.

Thành phần **Disconnected**

- ❖ **Disconnected** cho phép bất kỳ nguồn dữ liệu nào cũng có thể đưa vào **DataSet** hoặc **DataTable**.
- ❖ Tuy nhiên, **disconnected** không thể tự đưa dữ liệu vào được; phải tương tác với các thành phần của **Connected** thông qua **Data Adapter**.

Data Adapter

- ❖ Là thành phần cầu nối cho **Disconnected** với nguồn dữ liệu.
- ❖ Giúp đưa dữ liệu vào các thành phần của **Disconnected** và lưu trữ dữ liệu vào lại nguồn dữ liệu.
- ❖ Giúp Disconnected tương tác được với **Connected** để thực hiện các truy vấn dữ liệu.

Data Provider

- ❖ Cung cấp các class để các thành phần trong Connected làm việc với các nguồn dữ liệu khác nhau.
- ❖ VD: sử dụng cho SQL Server:

```
using System.Data.SqlClient;
```

Nguồn dữ liệu	Data Provider
Microsoft SQL Server	System.Data.SqlClient
MySQL	MySql.Data.MySqlClient
Oracle	System.Data.OracleClient
ODBC data source	System.Data.ODBC
OLE DB data source	System.Data.OleDb

Thông số kết nối CSDL

- ❖ Thông số kết nối sẽ thay đổi theo loại CSDL được kết nối.
- ❖ Sử dụng lớp `SqlConnection` để tạo ra đối tượng kết nối với SQL Server.
- ❖ Tạo đối tượng của lớp `SqlConnection` và cung cấp chuỗi kết nối vào thuộc tính `ConnectionString`.
- ❖ `ConnectionString`: là chuỗi ký tự chữ cặp khóa – giá trị, mỗi cặp là một tham số cho việc kết nối, cách nhau bởi “;”.
- ❖ Cung cấp các giá trị để kết nối: tên server, username, password, cách xác thực của server.
- ❖ Không phân biệt thứ tự, không phân biệt chữ hoa thường của các cặp tham số trong `ConnectionString`.

Thông số kết nối CSDL

- ❖ Các giá trị trong chuỗi kết nối trong **SQL Server**:
 - Xác định tên server (hostname, nơi SQL cài đặt) và tên instance (không bắt buộc) dùng từ khóa **Data Source** hoặc **Server**; giá trị tên server: **(local)**, **.** (dấu chấm), **localhost**; tên instance cách hostname bằng dấu **"\"**: **sqlexpress**
Data Source=. hoặc **.\sqlexpress**
Data Source=(local) hoặc **(local)\sqlexpress**
Data Source=localhost hoặc **localhost\sqlexpress**
 - Chỉ định CSDL: dùng từ khóa **Initial Catalog** và giá trị là tên CSDL muốn sử dụng.
Initial Catalog=Test

Thông số kết nối CSDL

- Xác thực CSDL: chuỗi kết nối cần chứa thông tin về xác thực CSDL. SQL Server có 2 loại: *windows authentication* và *SQL Server authentication*.
 - Windows authentication: dùng từ khóa **Integrated Security** với giá trị **true** hoặc **SSPI**
Integrated Security=True
Integrated Security=SSPI
 - SQL Server authentication: dùng từ khóa **User ID** và **Password** với giá trị khi cài đặt CSDL
User ID=sa
Password=123456

Thông số kết nối CSDL

- ❖ Chuỗi kết nối đầy đủ để kết nối với **SQL Server**

```
var conString = @"Data Source=localhost;Initial  
Catalog=Contacts;Integrated Security=True";
```

- ❖ Chuỗi kết nối đầy đủ để kết nối với **MySQL**

```
string cs = @"server=localhost;user id=root;password=;database=quanlycasi";
```

Thông số kết nối CSDL

- ❖ Sử dụng lớp `ConnectionStringBuilder` để tạo chuỗi kết nối.

```
SqlConnectionStringBuilder conStringBuilder = new  
SqlConnectionStringBuilder();  
conStringBuilder.DataSource = "localhost";  
conStringBuilder.InitialCatalog = "Contacts";  
conStringBuilder.IntegratedSecurity = true;
```

Thông số kết nối CSDL

- ❖ Lưu và truy xuất từ file cấu hình **App.config**.
 - File cấu hình App.config là một file xml, được tạo tự động khi tạo project.
 - Dùng để lưu trữ cấu hình của ứng dụng.

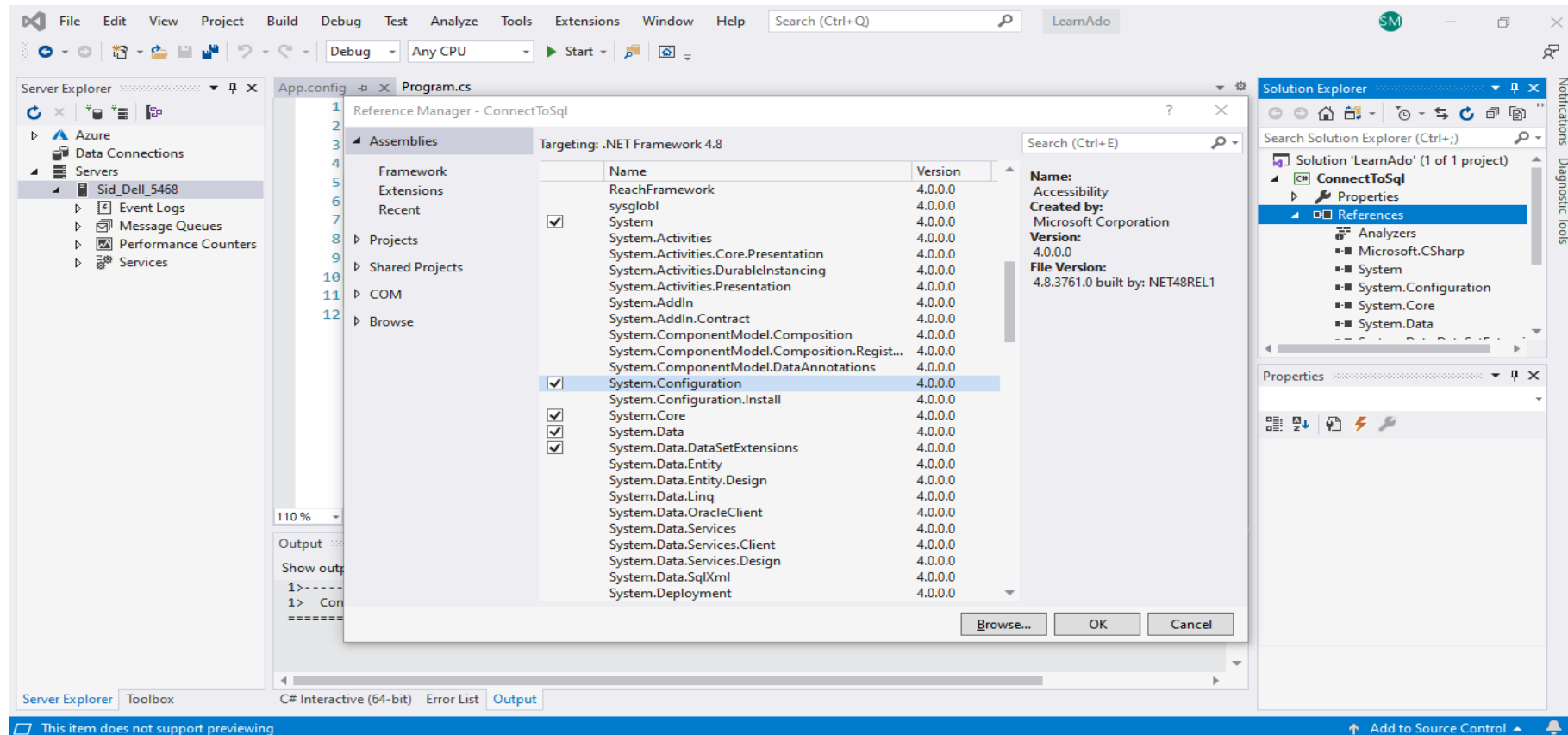
```
<?xml version="1.0" encoding="utf-8" ?>  
<configuration>  
  <startup>  
    <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.8" />  
  </startup>  
</configuration>
```

- Thêm node <connectionStrings> và node <add /> cùng các thuộc tính:

```
<connectionStrings>  
  <add name="my connection string" connectionString="Data Source=localhost  
    ;Initial Catalog=Contacts;Integrated Security=True" />  
</connectionStrings>
```

Thông số kết nối CSDL

- Thêm thư viện `System.Configuration` vào project:
click phải References => Add Reference => chọn `System.Configuration`.



Thông số kết nối CSDL

- Thêm thư viện `System.Configuration` vào code
 - `using System.Configuration;`
- Tạo chuỗi kết nối

```
var conString = ConfigurationManager.ConnectionStrings["my connection string"].ConnectionString;
```

Kết nối CSDL với SQL Server

❖ Tạo đối tượng của lớp `SqlConnection` và cung cấp chuỗi kết nối vào thuộc tính `ConnectionString`.

○ Cách 1:

```
SqlConnection conn = new SqlConnection();  
conn.ConnectionString = connString;
```

○ Cách 2:

```
SqlConnection conn = new SqlConnection(connString);
```

○ Cách 3:

```
SqlConnection conn = new SqlConnection{  
    ConnectionString = connString  
};
```

Kết nối CSDL với MySQL

❖ Tạo đối tượng của lớp `MySqlConnection` và cung cấp chuỗi kết nối vào thuộc tính `ConnectionString`.

○ Cách 1:

```
MySqlConnection conn = new MySqlConnection();  
conn.ConnectionString = connString;
```

○ Cách 2:

```
MySqlConnection conn = new MySqlConnection(connString);
```

○ Cách 3:

```
MySqlConnection conn = new MySqlConnection{  
    ConnectionString = connString  
};
```


Kết nối CSDL

- ❖ Phương thức `Open()`: mở kết nối tới CSDL.

```
connection.Open();
```

- ❖ Phương thức `Close()`: đóng kết nối CSDL; tuy nhiên kết nối vẫn chưa thực sự xóa bỏ mà được đưa vào `Connection Pool` để tái sử dụng.

```
connection.Close();
```

Phương thức `Dispose()`: đóng kết nối CSDL đồng thời xóa bỏ tất cả trạng thái và thông tin của `connection` (như Connection String) và chương trình không thể tái sử dụng.

```
connection.Dispose();
```

Kết nối CSDL

❖ Một số thuộc tính và phương thức của lớp `SqlConnection`

○ `int ConnectionTimeout { get; }:`

Thời gian tối đa để kết nối tới server (tính bằng giây, mặc định là 15 giây).

Sau thời gian này nếu ko kết nối được sẽ báo lỗi.

Sử dụng tham số `Connection Timeout` trong chuỗi kết nối hoặc thuộc tính `ConnectTimeout` của `SqlConnectionStringBuilder` để thiết lập

○ `string Database { get; }:` lấy các thông số của CSDL đang sử dụng

○ `ConnectionState State { get; }:`

Lấy thông tin về trạng thái hiện tại của kết nối.

Giá trị thuộc kiểu `ConnectionState` (thuộc `System.Data`).

Hai giá trị thường sử dụng: `ConnectionState.Open` và `ConnectionState.Closed`

Kết nối CSDL

- `ChangeDatabase` (`string` database): chuyển đổi sang CSDL khác với CSDL đã được thiết lập trước đó.
- `SqlCommand` `CreateCommand` (): tạo ra một object của lớp `SqlCommand` để thực hiện các truy vấn trên CSDL.

Connection Pooling

- ❖ Là kỹ thuật cho phép tạo và duy trì một số kết nối sử dụng chung để tăng hiệu suất cho ứng dụng.
- ❖ Thực hiện thông qua tái sử dụng kết nối khi có yêu cầu mà không phải tạo kết nối mới.
- ❖ Do việc tạo và hủy kết nối thường mất nhiều thời gian và việc đóng mở kết nối liên tục sẽ ảnh hưởng đến hiệu năng của ứng dụng và chịu tải của server.
- ❖ Trong ADO.NET, Connection Pooling được sử dụng mặc định.
- ❖ Trong SqlConnection, khi gọi phương thức Close hoặc Dispose kết nối sẽ tự động đưa vào vùng lưu trữ tạm (pool) và ko bị hủy hoàn toàn.
- ❖ Khi gọi lệnh Open tiếp theo, kết nối trong pool sẽ được sử dụng.

Ví dụ mở connection

```
var conString = @"Data Source=localhost;Initial Catalog=Contacts;Integrated Security=True";
var connection = new SqlConnection
{
    ConnectionString = conString
};
try
{
    connection.Open();
    if (connection.State == ConnectionState.Open)
    {
        Console.WriteLine("Connection opened successfully!");
    }
}
catch (Exception e)
{
    if (connection.State != ConnectionState.Open)
    {
        Console.WriteLine("Failed to open the connection");
        Console.WriteLine(e.ToString());
    }
}
finally
{
    connection.Close();
}
```

Tạo câu lệnh SQL và thực thi

- ❖ Trong ADO.NET sử dụng lớp `SqlCommand` để thực thi các câu truy vấn SQL và xử lý kết quả trả về.
- ❖ Thuộc thư viện `System.Data.SqlClient`.
- ❖ Khai báo và khởi tạo lớp `SqlCommand`
 - **Cách 1:** khởi tạo đối tượng bình thường

```
var command = new SqlCommand();  
var queryString = "Select * from Test";  
command.CommandText = queryString;  
command.Connection = connection;
```
 - **Cách 2:** cung cấp câu lệnh truy vấn và khởi tạo

```
var queryString = "Select * from Test";  
var command = new SqlCommand(queryString);  
command.Connection = connection;
```

Tạo câu lệnh SQL và thực thi

- Cách 3: cung cấp câu lệnh truy vấn và connection

```
var queryString = "Select * from Test";
```

```
var command = new SqlCommand(queryString, connection);
```

- Cách 4: Tạo trực tiếp lệnh command từ connection

```
var command = connection.CreateCommand();
```

```
var queryString = "Select * from Test";
```

```
command.CommandText = queryString;
```

Tạo câu lệnh SQL và thực thi

- 📖 Một số thuộc tính và phương thức của `SqlCommand`
 - `string CommandText` { `get`; `set`; }: chứa câu truy vấn SQL
 - `SqlConnection Connection` { `get`; `set`; }: chứa đối tượng connection để kết nối CSDL.
 - `CommandType CommandType` { `get`; `set`; }: chỉ ra thực hiện câu truy vấn SQL (`CommandType.Text`) hoặc gọi hàm stored procedure (`CommandType.StoredProcedure`); mặc định là `Text`.
 - `SqlParameterCollection Parameters` { `get`; }: danh sách tham số được sử dụng trong truy vấn.
 - `int ExecuteNonQuery` (): thực thi các câu truy vấn thêm, xóa và sửa (không trả dữ liệu về).
 - `object ExecuteScalar` (): thực thi các câu truy vấn có hàm tính toán (`Select COUNT|MIN|MAX|AVG`)
 - `SqlDataReader ExecuteReader` (): thực thi câu truy vấn Select

Thực thi câu truy vấn Select

```
var conString = @"Data Source=localhost;Initial Catalog=QLBH;Integrated Security=True";
var connection = new SqlConnection(conString);

var queryString = "Select * from KHACHHANG";
var command = new SqlCommand(queryString, connection);
connection.Open();
var reader = command.ExecuteReader(CommandBehavior.CloseConnection);
if (reader.HasRows)
{
    while (reader.Read())
    {
        var makh = reader.GetString(0);
        var hoTen = reader.GetString(1);
        var sdt = reader.GetString(3);
        Console.WriteLine($"MAKH: {makh}\t họ tên: {hoTen}\t điện thoại: {sdt}");
    }
}

connection.Close();
```

Thực thi câu truy vấn Select

- ❖ Sử dụng phương thức `ExecuteReader` để thực thi câu truy vấn select.
 - ❖ Xử lý kết quả trả về với `SqlDataReader`:
 - Kết quả trả về là object của `SqlDataReader`.
 - Đọc dữ liệu theo chiều từ đầu đến cuối (forward-only) và không được sửa (read-only).
 - `bool HasRows { get; }`: kiểm tra truy vấn có trả về dữ liệu.
 - Dùng `Read()` để duyệt lần lượt các dòng dữ liệu.
 - Dùng `GetXxx(index)`: để đọc dữ liệu trong từng ô dữ liệu:
 - `Xxx`: tương ứng với kiểu dữ liệu của ô
 - `Index`: số thứ tự của ô trong mỗi dòng.
- ```
var makh = reader.GetInt32(0);
var hoTen = reader.GetString(1);
```

# Thực thi câu truy vấn Select

- Dùng **chỉ số index** hoặc **tên cột dữ liệu** để đọc dữ liệu trong từng ô dữ liệu:

```
var hoTen = reader[1] as string;
```

```
var sdt = reader["SODT"] as string;
```

## ❖ Lưu ý:

- Mỗi truy vấn Select trả về dữ liệu thuộc một bảng (theo câu truy vấn) gọi là tập kết quả (result set).
- SqlCommand cho phép thực thi nhiều truy vấn cùng lúc. Khi đó ExecuteReader sẽ trả về nhiều tập kết quả.

```
var queryString = "Select * from KHACHHANG; Select * from
NHANVIEN";
```

- Sử dụng **NextResult ()** để chuyển sang result set kế tiếp .
- Sau khi hoàn thành đọc dữ liệu cần đóng SqlDataReader nếu không các câu truy vấn tiếp theo sẽ không được thực thi.

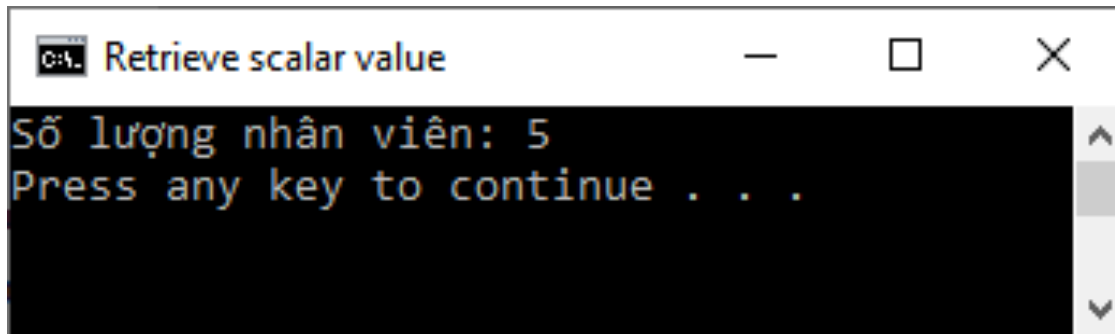
# Thực thi câu truy vấn Select

- ❖ **CommandBehavior**: tham số của phương thức ExecuteReader để xác định các hành động khi thực thi truy vấn.
- ❖ **Các giá trị của CommandBehavior**:
  - **Default**: truy vấn trả về nhiều tập kết quả; mặc định không gần ghi rõ.
  - **SingleResult**: truy vấn trả về một tập kết quả.
  - **SingleRow**: truy vấn trả về một dòng dữ liệu.
  - **KeyInfo**: truy vấn chỉ lấy thông tin về cột và khóa chính
  - **SchemaOnly**: Truy vấn trả về thông tin của cột, không có dữ liệu
  - **CloseConnection**: SqlDataReader sẽ đóng sau khi hoàn thành đọc dữ liệu

# Thực thi với ExecuteScalar

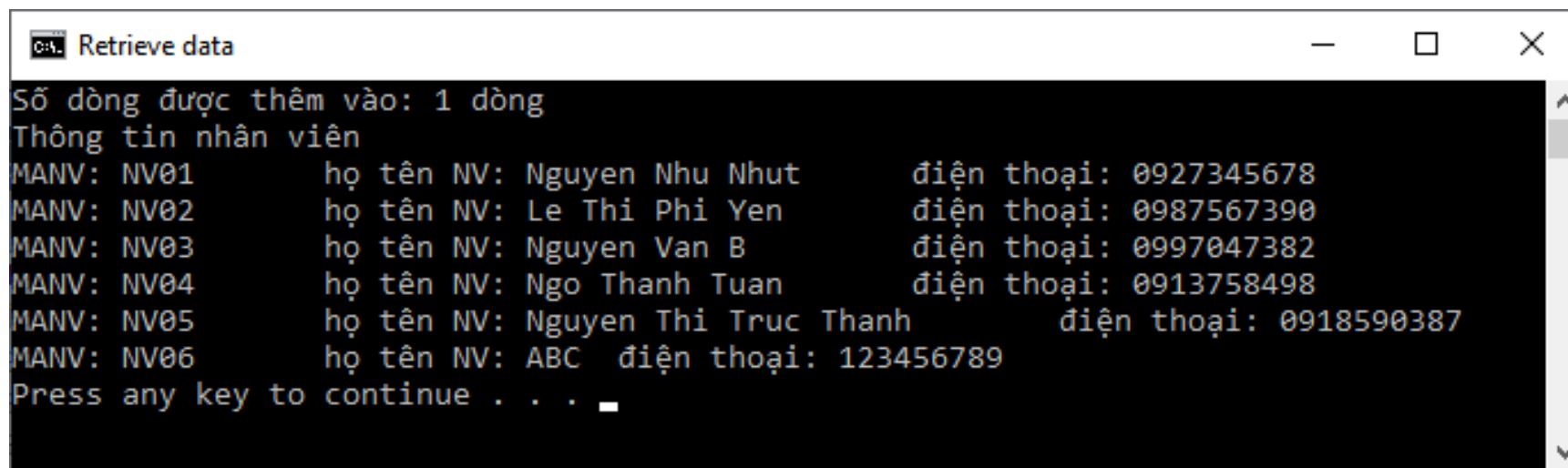
- ❖ **ExecuteScalar**: được dùng thực thi câu truy vấn các hàm tính toán (Aggregate), khi câu lệnh truy vấn trả về một giá trị
- ❖ Giá trị trả về là object, cần ép kiểu để sử dụng.

```
Console.Title = "Retrieve scalar value";
Console.OutputEncoding = Encoding.UTF8;
var conString = @"Data Source=localhost;Initial Catalog=QLBH;Integrated Security=True";
using (var connection = new SqlConnection(conString))
{
 var queryString = "Select count(*) from NHANVIEN";
 var command = new SqlCommand(queryString, connection);
 connection.Open();
 var count = (int)command.ExecuteScalar();
 Console.WriteLine($"Số lượng nhân viên: {count}");
}
```

A screenshot of a Windows console window. The title bar reads "C:\. Retrieve scalar value". The console output shows "Số lượng nhân viên: 5" on the first line and "Press any key to continue . . ." on the second line. The cursor is positioned at the end of the second line.

# Thực thi câu truy vấn Insert

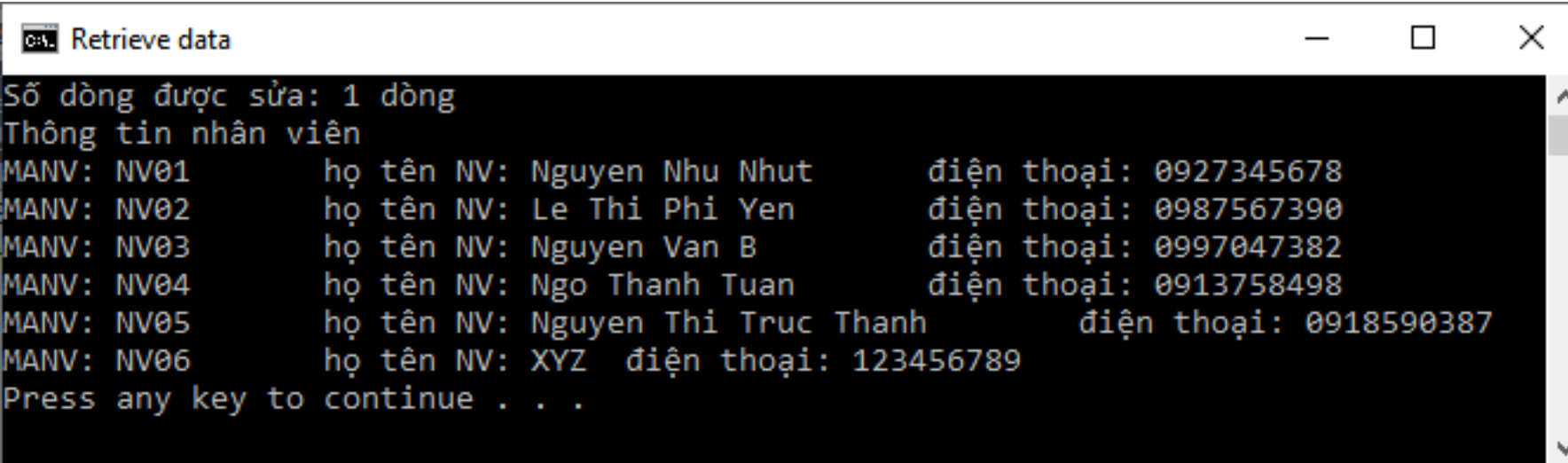
```
var conString = @"Data Source=localhost;Initial Catalog=QLBH;Integrated Security=True";
using (var connection = new SqlConnection(conString))
using (var command = new SqlCommand { Connection = connection })
{
 var insertString = "Insert into NHANVIEN(MANV, HOTEN, SODT, NGVL) " +
 "Values (N'NV06', N'ABC', N'123456789', N'2020-01-01') ";
 command.CommandText = insertString;
 connection.Open();
 var count = command.ExecuteNonQuery();
 Console.WriteLine($"Số dòng được thêm vào: {count} dòng");
}
```



```
Retrieve data
Số dòng được thêm vào: 1 dòng
Thông tin nhân viên
MANV: NV01 họ tên NV: Nguyen Nhu Nhut điện thoại: 0927345678
MANV: NV02 họ tên NV: Le Thi Phi Yen điện thoại: 0987567390
MANV: NV03 họ tên NV: Nguyen Van B điện thoại: 0997047382
MANV: NV04 họ tên NV: Ngo Thanh Tuan điện thoại: 0913758498
MANV: NV05 họ tên NV: Nguyen Thi Truc Thanh điện thoại: 0918590387
MANV: NV06 họ tên NV: ABC điện thoại: 123456789
Press any key to continue . . .
```

# Thực thi câu truy vấn Update

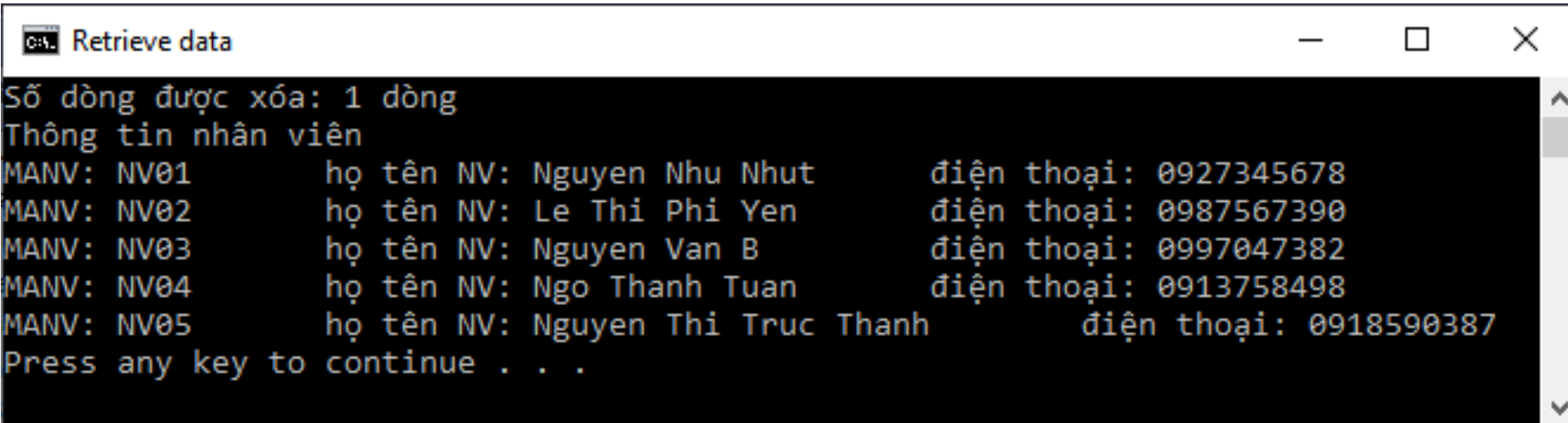
```
var conString = @"Data Source=localhost;Initial Catalog=QLBH;Integrated Security=True";
using (var connection = new SqlConnection(conString))
using (var command = new SqlCommand { Connection = connection })
{
 var updateString = "Update NHANVIEN set HOTEN = 'XYZ' where MANV = 'NV06'";
 command.CommandText = updateString;
 connection.Open();
 var count = command.ExecuteNonQuery();
 Console.WriteLine($"Số dòng được sửa: {count} dòng");
}
```



```
C:\> Retrieve data
Số dòng được sửa: 1 dòng
Thông tin nhân viên
MANV: NV01 họ tên NV: Nguyen Nhu Nhut điện thoại: 0927345678
MANV: NV02 họ tên NV: Le Thi Phi Yen điện thoại: 0987567390
MANV: NV03 họ tên NV: Nguyen Van B điện thoại: 0997047382
MANV: NV04 họ tên NV: Ngo Thanh Tuan điện thoại: 0913758498
MANV: NV05 họ tên NV: Nguyen Thi Truc Thanh điện thoại: 0918590387
MANV: NV06 họ tên NV: XYZ điện thoại: 123456789
Press any key to continue . . .
```

# Thực thi câu truy vấn Delete

```
var conString = @"Data Source=localhost;Initial Catalog=QLBH;Integrated Security=True";
using (var connection = new SqlConnection(conString))
using (var command = new SqlCommand { Connection = connection })
{
 var updateString = "Delete from NHANVIEN where MANV = 'NV06'";
 command.CommandText = updateString;
 connection.Open();
 var count = command.ExecuteNonQuery();
 Console.WriteLine($"Số dòng được xóa: {count} dòng");
}
```



```
C:\> Retrieve data
Số dòng được xóa: 1 dòng
Thông tin nhân viên
MANV: NV01 họ tên NV: Nguyen Nhu Nhut điện thoại: 0927345678
MANV: NV02 họ tên NV: Le Thi Phi Yen điện thoại: 0987567390
MANV: NV03 họ tên NV: Nguyen Van B điện thoại: 0997047382
MANV: NV04 họ tên NV: Ngo Thanh Tuan điện thoại: 0913758498
MANV: NV05 họ tên NV: Nguyen Thi Truc Thanh điện thoại: 0918590387
Press any key to continue . . .
```



# Tham số trong câu truy vấn

**NO!**

```
var ma = Console.ReadLine();
var hoten = Console.ReadLine();
var sdt = Console.ReadLine();
var ns = Console.ReadLine();

var insertString = $"Insert into NHANVIEN(MANV, HOTEN, SODT, NGVL)" +
 $" values ('{ma}', '{hoten}', '{sdt}', '{ns}')";
```

# Tham số trong câu truy vấn

- ❖ Lớp `SqlParameter`: được dùng để truyền các tham số vào câu truy vấn.
- ❖ Trong câu truy vấn đặt tham số ở chỗ cần thêm, bắt đầu bằng ký tự @ (@<tên tham số>)

```
var queryString = "Select * from NHANVIEN where MANV = @MANV";
```

- ❖ Khai báo các object của `SqlParameter` và gán trị.

```
var manv = new SqlParameter("MANV", ma);
```

- ❖ Gán object `SqlParameter` vào thuộc tính `Parameters` của `SqlCommand`.

```
command.Parameters.Add(manv);
```

- ❖ Hoặc có thể thêm bằng phương thức `AddWithValue()` của `SqlCommand`

```
command.Parameters.AddWithValue("MANV", ma);
```

# Tham số trong câu truy vấn

```
var queryString = "Select * from NHANVIEN where MANV = @MANV";
Console.Write("Nhập mã nhân viên muốn tìm: ");
var ma = Console.ReadLine();
using (var connection = new SqlConnection(connectionString))
using (var command = new SqlCommand { Connection = connection })
{
 command.CommandText = queryString;
 //Thêm bằng AddWithValue
 command.Parameters.AddWithValue("MANV", ma);

 //Thêm bằng object SqlParameter
 //var idNV = new SqlParameter("MANV", ma);
 //command.Parameters.Add(idNV);

 connection.Open();
 var reader = command.ExecuteReader(CommandBehavior.CloseConnection);
 if (reader.HasRows)
 {
 Console.WriteLine("Thông tin nhân viên:");
 while (reader.Read())
 {
 var manv = reader.GetString(0);
 var hoTen = reader[1] as string;
 var sdt = reader["SODT"] as string;
 Console.WriteLine($"MANV: {manv}\t họ tên NV: {hoTen}\t điện thoại: {sdt}");
 }
 }
}
```

# Tham số trong câu truy vấn

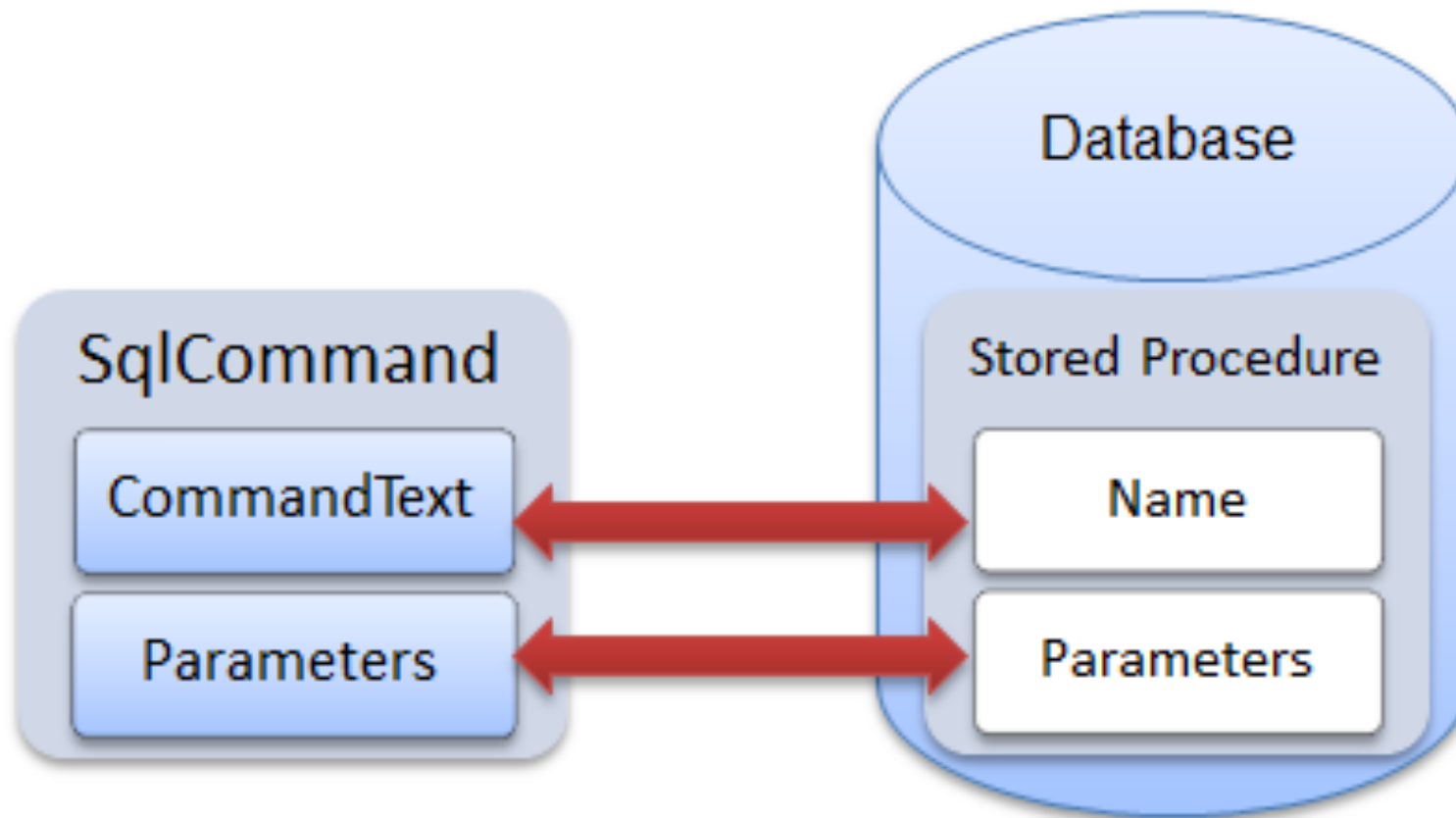
```
Console.Write("Thêm mã nhân viên: ");
var ma = Console.ReadLine();
Console.Write("Thêm họ tên nhân viên: ");
var hoTen = Console.ReadLine();
Console.Write("Thêm số điện thoại: ");
var dt = Console.ReadLine();
Console.Write("Thêm ngày sinh: ");
var ns = Console.ReadLine();

var insertString = "Insert into NHANVIEN(MANV, HOTEN, SODT, NGVL) " +
 "Values (@MANV, @HOTEN, @SDT, @NS) ";

var manv = new SqlParameter("MANV", ma);
var ten = new SqlParameter("HOTEN", hoTen);
var sdt = new SqlParameter("SDT", dt);
var ngs = new SqlParameter("NS", ns);

using (var connection = new SqlConnection(connectionString))
using (var command = new SqlCommand { Connection = connection })
{
 command.CommandText = insertString;
 command.Parameters.Add(manv);
 command.Parameters.Add(ten);
 command.Parameters.Add(sdt);
 command.Parameters.Add(ngs);
 connection.Open();
 var count = command.ExecuteNonQuery();
 Console.WriteLine($"Số dòng được thêm vào: {count} dòng");
}
```

# Thực thi với Stored Procedure



# Thực thi với Stored Procedure

```
var conString = @"Data Source=localhost;Initial Catalog=QLBH;Integrated Security=True";
using (var connection = new SqlConnection(conString))
using (var command = new SqlCommand { Connection = connection })
{
 command.CommandType = CommandType.StoredProcedure;
 command.CommandText = "GetList";
 connection.Open();
 var reader = command.ExecuteReader(CommandBehavior.CloseConnection);
 if (reader.HasRows)
 {
 Console.WriteLine("Thông tin nhân viên:");
 while (reader.Read())
 {
 var manv = reader.GetString(0);
 var hoTen = reader[1] as string;
 var sdt = reader["SODT"] as string;
 Console.WriteLine($"MANV: {manv}\t họ tên NV: {hoTen}\t điện thoại: {sdt}");
 }
 }
}
```

# Thực thi với Stored Procedure

```
var conString = @"Data Source=localhost;Initial Catalog=QLBH;Integrated Security=True";
Console.Write("Nhập mã khách hàng: ");
var ma = Console.ReadLine();
using (var connection = new SqlConnection(conString))
using (var command = new SqlCommand { Connection = connection })
{
 command.CommandType = CommandType.StoredProcedure;
 command.CommandText = "GetListParam";
 command.Parameters.AddWithValue("MAKH", ma);
 connection.Open();
 var reader = command.ExecuteReader(CommandBehavior.CloseConnection);
 if (reader.HasRows)
 {
 Console.WriteLine("Thông tin khách hàng:");
 while (reader.Read())
 {
 var makh = reader.GetString(0);
 var hoTen = reader[1] as string;
 var sdt = reader["SODT"] as string;
 Console.WriteLine($"MAKH: {makh}\t họ tên: {hoTen}\t điện thoại: {sdt}");
 }
 }
}
```

# Tương quan dữ liệu MySQL với C#

| MySQL             | C#       |
|-------------------|----------|
| char(x)           | string   |
| datetime          | DateTime |
| varchar(x)        | string   |
| smallint          | short    |
| smallint unsigned | ushort   |
| int               | int      |
| int unsigned      | uint     |
| bigint            | long     |
| tinyint           | sbyte    |
| bigint unsigned   | ulong    |
| text              | string   |



# Bài tập

---

- `casi(MaCaSi, TenCaSi, GioiTinh)`
    - `MaCaSi, TenCaSi`: chuỗi 50 ký tự, `GioiTinh` (bool)
  - `album(MaAlBum, TenAlbum, MaCaSi)`
    - Tất cả đều là chuỗi
  - `baihat(MaBaiHat, TenBaiHat, TheLoai, MaAlBum)`
    - Tất cả chuỗi
- 1 ca sỹ có nhiều album, một album có nhiều bài hát

# Bài tập

---

- `casi(MaCaSi, TenCaSi, GioiTinh)`
    - `MaCaSi, TenCaSi`: chuỗi 50 ký tự, `GioiTinh` (bool)
  - `album(MaAlBum, TenAlbum, MaCaSi)`
    - Tất cả đều là chuỗi
  - `baihat(MaBaiHat, TenBaiHat, TheLoai, MaAlBum)`
    - Tất cả chuỗi
- 1 ca sỹ có nhiều album, một album có nhiều bài hát

# Bài tập

---

1. Hàm thêm và gọi hàm để thêm 3 ca sĩ, mỗi ca sĩ thêm 3 album

- Thêm ca sĩ
- Thêm album

2. Liệt kê

- Liệt kê tên ca sĩ, giới tính của ca sĩ
- Liệt kê những ca sĩ có giới tính nam
- thông tin ca sĩ, cùng với số lượng album của từng ca sĩ

3. Xóa Album theo mã album

4. Xóa ca sĩ theo mã số



# LINQ

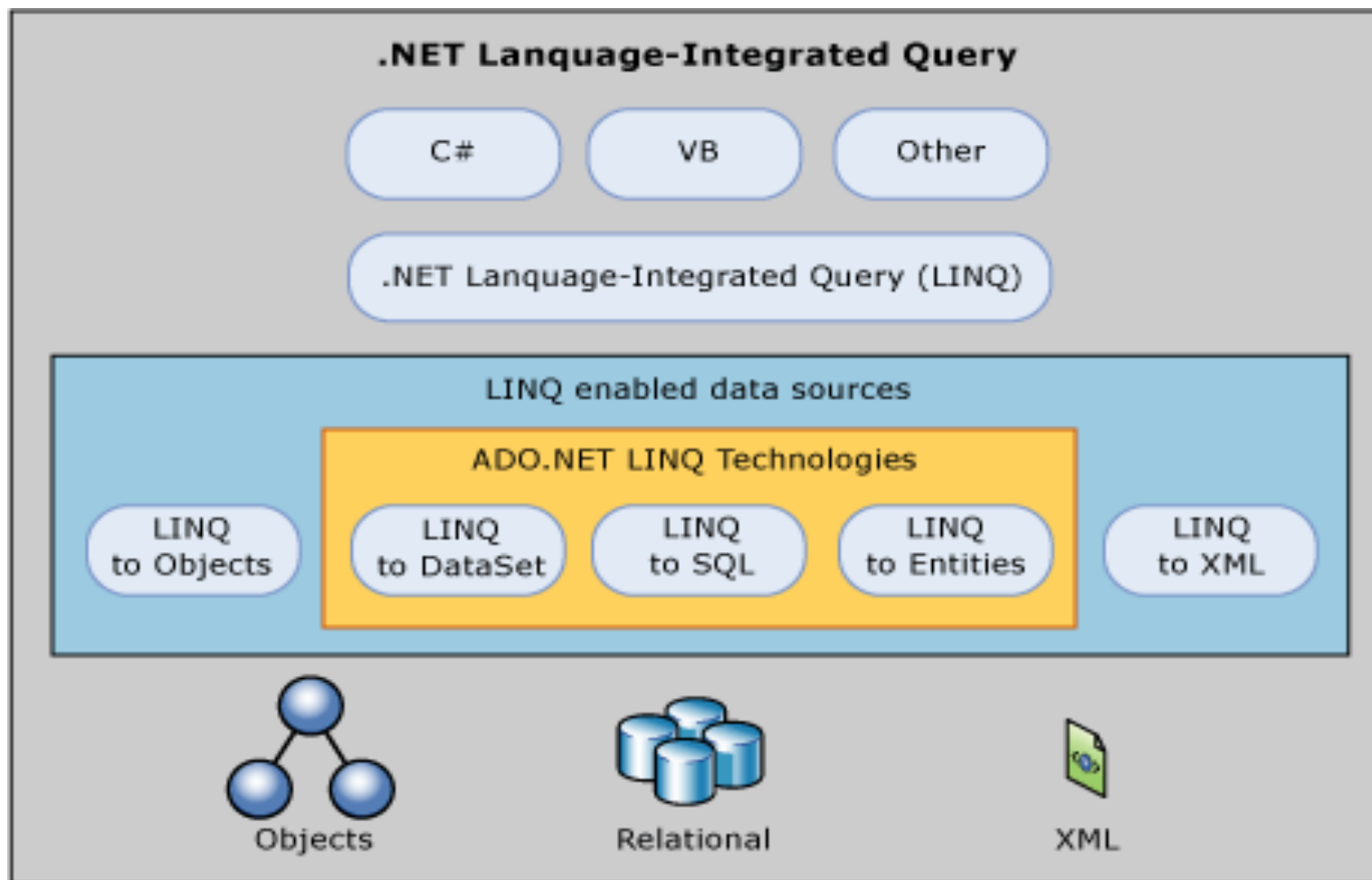
# Language-Integrated Query

- ❖ Là ngôn ngữ truy vấn dữ liệu chung cho các ngôn ngữ lập trình của .NET Framework.
- ❖ Được dùng cho truy vấn chung trên nhiều nguồn dữ liệu khác nhau (SQL Server, XML...)
- ❖ Cú pháp gần giống với SQL và cú pháp Method
- ❖ Tích hợp vào ngôn ngữ của .NET và loại bỏ được sự khác biệt giữa ngôn ngữ lập trình với ngôn ngữ truy vấn dữ liệu.
- ❖ Các kiểu của LINQ:
  - *LINQ to Objects*: truy vấn đến các đối tượng.
  - *LINQ to XML*: truy vấn đến dữ liệu XML
  - *LINQ to ADO.NET*: làm việc với ADO.NET, cho phép truy vấn đến DataSet (*LINQ to DataSet*), chuyển thành các lệnh SQL (*LINQ to SQL*), truy vấn đến thực thể trong Entity Framework (*LINQ to Entities*)

# Language-Integrated Query

- ❖ Tất cả các **class** và **interface** cho LINQ đều thuộc thư viện **System.Linq**.
- ❖ Biểu thức truy vấn tương tự trong SQL với các toán tử truy vấn: **From**, **Where...** và thường bắt đầu bằng **From**.
- ❖ Thuộc thư viện **System.Query**.
- ❖ *Ưu điểm:*
  - Dễ xử lý lỗi trong khi thiết kế
  - Viết code nhanh, gần gũi với cách viết của ngôn ngữ trong .NET Framework.
  - Câu truy vấn dễ hiểu, ngắn gọn để tương tác và xử lý dữ liệu
  - Dễ debug với .NET Debugger.
  - Truy vấn trên nhiều nguồn dữ liệu với một cú pháp chung.
  - Dễ dàng chuyển đổi từ kiểu dữ liệu này sang kiểu dữ liệu khác.

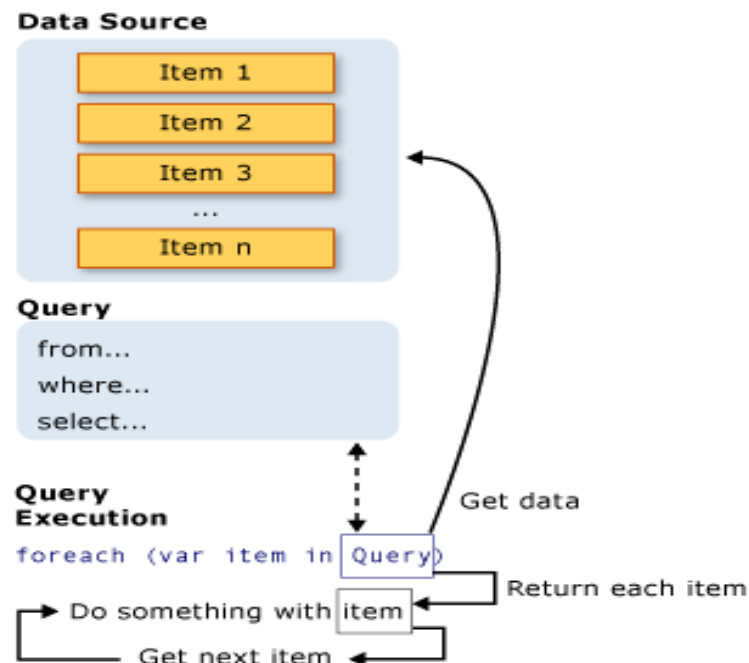
# Kiến trúc của LINQ



<https://docs.microsoft.com/en-us/dotnet/framework/data/adonet/linq-and-ado-net?redirectedfrom=MSDN>

# Truy vấn trong LINQ

- ❖ Truy vấn trong LINQ gồm 3 phần:
  - *Nguồn dữ liệu* (data source): mảng, danh sách, XML, database...
  - *Truy vấn* (query): các biểu thức truy vấn.
  - *Thực thi truy vấn* (query execution): hiển thị kết quả truy vấn.



<https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/linq/introduction-to-linq-queries>



# Truy vấn trong LINQ

---

## ➤ Nguồn dữ liệu (data source):

- ❖ LINQ dùng chung cho nhiều nguồn dữ liệu nên luôn làm việc với Object.
- ❖ Với mỗi loại nguồn dữ liệu có LINQ Provider riêng để chuyển đổi dữ liệu về dạng object và ngược lại: LINQ to XML cho XML, LINQ to SQL cho CSDL SQL Server
- ❖ Phương thức mở rộng trong LINQ thuộc các class **Enumerable** và **Queryable**
- ❖ Class **Enumerable** chứa các phương thức thực thi giao diện **IEnumerable**
- ❖ Class **Queryable** chứa các phương thức thực thi giao diện **IQueryable**

# Truy vấn trong LINQ

## ➤ Truy vấn(query):

- ❖ *Cú pháp truy vấn ([query syntax](#))* : đảo ngược cách viết truy vấn select trong SQL; đưa **from** lên đầu và kết thúc là **select**.

```
IEnumerable<int> listEvenNumber1 = from num in numbers
 where num % 2 == 0
 orderby num
 select num;
```

- ❖ *Cú pháp phương thức ([method syntax](#))*: giống cách gọi một phương thức bình của object; là cách viết cơ bản của LINQ.

```
IEnumerable<int> listEvenNumber2 = numbers.Where(num => num % 2 ==
0).OrderBy(n => n);
```

- ❖ *Cú pháp pha trộn ([mixed syntax](#))*: một số phương thức LINQ không hỗ trợ ở dạng query syntax => sử dụng pha trộn 2 lối viết.

# Truy vấn trong LINQ

---

## ➤ Một số lưu ý

- ❖ Một số phương thức chỉ được viết theo **method syntax**, không viết được bằng **query syntax**.
- ❖ **Query syntax** được chuyển sang **method syntax** ở giai đoạn biên dịch; vì vậy, hai cách viết không khác biệt về hiệu suất thực thi.
- ❖ **Method syntax** thường viết theo kiểu biểu thức **lambda**.

# Truy vấn trong LINQ

## ➤ Danh sách phương thức truy vấn

| Chức năng               | Phương thức                                                     |
|-------------------------|-----------------------------------------------------------------|
| Lọc (Filtering)         | Where, OfType                                                   |
| Chiếu (Projection)      | Select, SelectMany                                              |
| Kết (Join)              | Join, GroupJoin                                                 |
| Sắp xếp (Sorting)       | OrderBy, OrderBy Descending, Reverse, ThenBy, ThenByDescending, |
| Nhóm (Grouping)         | GroupBy, ToLookup                                               |
| Kết gộp (Aggregation)   | Aggregate, Average, Count, LongCount, Max, Min, Sum             |
| Định lượng (Quantifier) | All, Any, Contains                                              |
| Tập hợp (Set)           | Distinct, Except, Intersection, Union                           |

# Truy vấn trong LINQ

| Chức năng                     | Phương thức                                                                                        |
|-------------------------------|----------------------------------------------------------------------------------------------------|
| Phần tử (Element)             | ElementAt, ElementAtOrDefault, First, FirstOrDefault, Last, LastOrDefault, Single, SingleOrDefault |
| Phân vùng dữ liệu (Partition) | Skip, SkipWhile, Take, TakeWhile                                                                   |
| Sinh dữ liệu (Generation)     | DefaultEmpty, Empty, Range, Repeat                                                                 |
| Chuyển kiểu (Conversions)     | AsEnumerable, AsQueryable, Cast, ToArray, ToDictionary, ToList                                     |
| Bằng nhau (Equality)          | SequenceEqual                                                                                      |
| Kết hợp (Concatenation)       | Concat                                                                                             |

# Truy vấn trong LINQ

## ➤ Thực thi truy vấn

### ❖ Thực thi trì hoãn (deferred execution):

- Truy vấn LINQ chỉ được thực thi khi cần **đến dữ liệu từ truy vấn đó** (khi có câu lệnh duyệt danh sách kết quả).
- Kết quả truy vấn có thể thay đổi trước khi lệnh duyệt dữ liệu.
- Tăng hiệu suất xử lý vì hạn chế thực thi những lệnh chưa cần thiết.
- Là đặc trưng rất mạnh trong LINQ và là cách thực thi mặc định.
- Áp dụng được cho tất cả các nguồn dữ liệu hỗ trợ LINQ.

# Truy vấn trong LINQ

```
//Tạo nguồn dữ liệu
int[] numbers = { 5, 4, 1, 3, 9, 8, 6, 7, 2, 0 };
//tạo truy vấn bằng LINQ
var lowNums = (from n in numbers
 where(n > 5)
 select n);
//Thực thi truy vấn
foreach (var x in lowNums)
{
 Console.WriteLine(x);
}
```

# Truy vấn trong LINQ

---

## Thực thi truy vấn:

### ❖ Thực thi ngay lập tức (forcing immediate execution):

Thực thi truy vấn ngay tại vị trí câu lệnh.

Kết quả không thay đổi sau khi lệnh tạo truy vấn hoàn tất.

Sử dụng các phương thức biến đổi dữ liệu bắt đầu bằng “To”: **ToArray**, **ToList**... để thực hiện loại truy vấn này.



# Truy vấn trong LINQ

```
static void Main(string[] args)
{
 //Tạo ngu ồn dữ liệu
 string[] words = { "Hello", "LINQ", "world" };
 //tạo truy vấn bằng LINQ
 var shortWords = (from word in words
 where word.Length <=5
 select word) .ToArray();

 //Thực thi truy vấn
 foreach (string word in shortWords)
 {
 Console.WriteLine(word);
 }
}
```

# Biểu thức Lambda

- ❖ Biểu thức Lambda (Lambda Expression): làm một **hàm nặc danh** (anonymous function) dùng để tạo các kiểu **delegates** hay cây biểu thức (expression tree).
- ❖ Dùng để viết một hàm cục bộ có thể truyền các tham số hay trả giá trị của hàm gọi.
- ❖ Hữu ích cho viết các truy vấn LINQ.
- ❖ Cú pháp: **(parameters) => { statement }**
- ❖ Dấu **=>** gọi là go-to:
- ❖ **VD:** **(x, y) => x == y;**  
**(int x, string s) => s.Length > x;**
- ❖ **Tham số** của biểu thức Lambda có thể **rỗng**

# Biểu thức Lambda

```
static void Main(string[] args)
{
 int[] array = { 1, 3, 5, 2, 9, 5, 6 };
 foreach (int a in array.Where(x =>
 {
 if (x >= 5) return true;
 else return false;
 }
))) Console.WriteLine(a);
}
```

# Một số phương thức trong Cú pháp phương thức

- ❖ Phương thức **Where**: dùng để lọc dữ liệu theo yêu cầu.
- ❖ Trả về danh sách dữ liệu kiểu **IEnumerable<TSource>**
- ❖ Where có 2 overload với tham số truyền vào:
  - `Where<TSource> (Func<TSource, bool> predicate);`
  - `Where<TSource> (Func<TSource, int, bool> predicate);`
- ❖ Overload đầu tiếp nhận một biến thuộc kiểu delegate `Func<TSource, bool>`
- ❖ Over load thứ hai tương tự nhưng có thêm tham số kiểu `int` để chứa index của phần tử `Func<TSource, int, bool>`.
- ❖ Trong đó **Tsource** là kiểu dữ liệu cơ sở của nguồn dữ liệu.
- ❖ Kết quả dữ liệu trả về khi thỏa **true** và sẽ vào danh sách và truy xuất như mảng

# Ví dụ cho các phương thức trong LINQ

```
class People {
 private string name { get; set; }
 private int age { get; set; }
 private string country { get; set; }
 public string Name
 {
 set { name = value; }
 get { return name; }
 }
 public int Age
 {
 set { age = value; }
 get { return age; }
 }
}
```

# Ví dụ cho các phương thức trong LINQ

```
public string Country
{
 set { country = value; }
 get { return country; }
}

public People(string name, int age , string country) {
 this.name = name;
 this.age = age;
 this.country = country;
}

public People() { }
```

# Ví dụ cho các phương thức trong LINQ

```
static void Main(string[] args)
{
 Console.OutputEncoding = Encoding.UTF8;
 var peoples = new List<People>
 {
 new People{Name="Tung",Age = 20, Country ="Viet Nam"},
 new People{Name="Trung",Age = 19, Country ="Viet Nam"},
 new People{Name="Tuan",Age = 18, Country ="Mỹ"},
 new People{Name="Trinh",Age = 21, Country ="Anh"},
 new People{Name="Nha",Age = 24, Country ="Pháp"},
 };
}
```

# Một số phương thức trong LINQ

- ❖ VD: Liệt kê danh sách **people** ở “Viet Nam”
- ❖ Vị trí tham số có thể cung cấp bằng hàm lambda hoặc các loại phương thức.

```
var array = peoples
 .Where(d => d.Country == "Viet Nam");

foreach (var obj in array)
{
 Console.WriteLine(obj.Name+" "+obj.Age);
}
```



# Một số phương thức trong LINQ

- ❖ VD: Liệt kê danh sách **peoples** ở vị trí chẵn.
- ❖ Vị trí tham số cung cấp thêm **tham số đầu vào** là kiểu **int**

```
var a = peoples
 .Where((d, i) => i % 2 == 0);

foreach (var obj in a)
{
 Console.WriteLine(obj.Name + " " + obj.Age);
}
```

# Một số phương thức trong LINQ

- ❖ Phương thức **Select**: tương tự với truy vấn **Select** trong SQL.
- ❖ Thực hiện **chuyển đổi** dữ liệu từ **nguồn dữ liệu hình thức** này sang hình thức khác.
- ❖ **VD**: Bảng SinhVien lưu trữ thông tin sinh viên: Mã, tên, ngày sinh, email...; nhưng ta chỉ cần chọn vài thông tin trong đó
- ❖ Trả về danh sách dữ liệu kiểu **IEnumerable<TResult>**
- ❖ Select có 2 overload với tham số truyền vào:
  - **Select**<Tsource, TResult>(Func<TSource, TResult > selector);
  - **Select**<Tsource, TResult>(Func<TSource,**int**, TResult > selector);
- ❖ Trong đó **Tsource** là kiểu dữ liệu cơ sở của **nguồn** dữ liệu và **TResult** là kiểu giá trị trả lại bởi select

# Một số phương thức trong LINQ

- ❖ VD: Liệt kê danh sách **Name** và **Country** của **peoples**
- ❖ Kết quả trả về là mảng các chuỗi.

```
var info = peoples
 .Select(d => $" {d.Name}\t{d.Country}");
foreach (var obj in info)
{
 Console.WriteLine(obj);
}
```

```
var info = peoples.Select(d => new {d.Name, d.Country});
foreach (var obj in info)
{
 Console.WriteLine(obj.Name+ " " + obj.Country);
}
```

# Một số phương thức trong LINQ

- ❖ **VD:** Liệt kê danh sách **Name** và **Country** của **peoples** với **Age < 20**
- ❖ Lọc dữ liệu với **Where** rồi **Select**

```
var arrayb = peoples
 .Where(d => d.Age < 20)
 .Select(d => $"{d.Name}\t{d.Country}");
Console.WriteLine("Các sinh viên tuổi < 20");
foreach (var obj in arrayb)
{
 Console.WriteLine(obj);
}
```

# Cú pháp truy vấn Query Syntax

❖ **VD:** Liệt kê toàn bộ danh sách **peoples**

```
var aa = from p in peoples
 select p;
foreach (var obj in aa)
{
 Console.WriteLine(obj.Name + "\t" + obj.Age + "\t" + obj.Country);
}
```

❖ **VD:** Liệt kê danh sách **peoples** theo **Name**

```
var bb = from p in peoples
 select p.Name;
foreach (var obj in bb)
{
 Console.WriteLine(obj);
}
```

# Cú pháp truy vấn Query Syntax

- ❖ Lọc dữ liệu (filter) với mệnh đề **Where**
- ❖ **VD:** Liệt kê danh sách **peoples** ở “Viet Nam”

```
Console.WriteLine("Các người ở Việt Nam");
var cc = from p in peoples
 where p.Country == "Viet Nam"
 select p;
foreach (var obj in cc)
{
 Console.WriteLine(obj.Name + "\t" + obj.Age + "\t" + obj.Country);
}
```

# Cú pháp truy vấn Query Syntax

- ❖ Dùng các toán tử && (and), || (or) để bổ sung điều kiện lọc ở Where

```
var dd = peoples
 .Where(p => p.Country == "Viet Nam" && p.Age < 20);
```

```
Console.WriteLine("Các người ở Việt Nam và có tuổi < 20");
var dd = from p in peoples
 where p.Country == "Viet Nam" && p.Age < 20
 select p;
foreach (var obj in dd)
{
 Console.WriteLine(obj.Name + "\t" + obj.Age + "\t" + obj.Country);
}
```

# Cú pháp truy vấn Method Syntax

- ❖ Sắp xếp (order) với mệnh đề orderby (ascending - tăng, descending-giảm)
- ❖ VD: Sắp xếp danh sách **peoples** ở “Viet Nam” tăng theo Tuổi

```
var ee = peoples
 .Where(d => d .Country == "Viet Nam")
 .OrderBy(d => d.Age); //OrderByDescending sắp xếp giảm
foreach (var obj in ee)
{
 Console.WriteLine(obj.Name + "\t" + obj.Age + "\t" + obj.Country);
}
```



# Cú pháp truy vấn Query Syntax

- ❖ Sắp xếp (order) với mệnh đề orderby (ascending - tăng, descending-giảm)
- ❖ VD: Sắp xếp danh sách **peoples** ở “Viet Nam” tăng theo Tuổi

```
Console.WriteLine("Các người ở Việt Nam tăng theo tuổi");
var ee = from d in peoples
 where d.Country == "Viet Nam"
 orderby d.Age ascending
 select d;
foreach (var obj in ee)
{
 Console.WriteLine(obj.Name + "\t" + obj.Age + "\t" + obj.Country);
}
```

# Cú pháp truy vấn Method Syntax

- ❖ Gom nhóm kết quả theo **một từ khóa** bằng mệnh đề Group.
- ❖ Kết quả là một danh sách lồng một danh sách kiểu `IEnumerable<IGrouping<string, TResult>>`
- ❖ **VD:** Gom nhóm **peoples** theo **Country**

```
var gg = peoples
 .GroupBy(d => d.Country);
foreach (var namegroup in gg)
{
 Console.WriteLine(namegroup.Key);
 foreach (var obj in namegroup)
 Console.WriteLine(obj.Name + "\t" + obj.Age + "\t");
}
```

# Cú pháp truy vấn Query Syntax

- ❖ Gom nhóm kết quả theo **một từ khóa** bằng mệnh đề Group.
- ❖ Kết quả là một danh sách lồng một danh sách kiểu `IEnumerable<IGrouping<string, TResult>>`
- ❖ **VD:** Gom nhóm **peoples** theo **Country**

```
Console.WriteLine("Gom theo quốc gia");
var gg = from d in peoples
 group d by d.Country;
foreach (var namegroup in gg)
{
 Console.WriteLine(namegroup.Key);
 foreach (var obj in namegroup)
 Console.WriteLine(obj.Name + "\t" + obj.Age + "\t");
}
```

# Cú pháp truy vấn Method Syntax

❖ **VD:** Gom nhóm **peoples** theo **Country** và lấy số lượng là hơn 2

```
Console.WriteLine("Gom theo quốc gia lấy nhóm nào có số lượng >=2");
var ggthen2 = peoples
 .GroupBy(d => d.Country)
 .Where(gr => gr.Count() > 1);

foreach (var namegroup in ggthen2)
{
 Console.WriteLine(namegroup.Key);
 foreach (var obj in namegroup)
 Console.WriteLine(obj.Name + "\t" + obj.Age + "\t");
}
```

# Cú pháp truy vấn Query Syntax

❖ **VD:** Gom nhóm **peoples** theo **Country** và lấy số lượng là hơn 2

```
Console.WriteLine("Gom theo quốc gia lấy nhóm nào có số lượng >=2");
var ggthen2 = from d in peoples
 group d by d.Country into ggthen1
 where ggthen1.Count() > 1
 select ggthen1;

foreach (var namegroup in ggthen2)
{
 Console.WriteLine(namegroup.Key);
 foreach (var obj in namegroup)
 Console.WriteLine(obj.Name + "\t" + obj.Age + "\t");
}
```

# Cú pháp truy vấn method Syntax

❖ **VD:** Gom nhóm **peoples** theo **Country** và lấy số lượng là  $\geq 2$  và sắp xếp theo tuổi sau khi gom

```
var ggthen2 = peoples
 .GroupBy(d => d.Country)
 .Where(gr => gr.Count() > 1)
 .Select(std => new
 {
 Key = std.Key,
 peos = std.OrderBy(x => x.Age)
 });
foreach (var namegroup in ggthen2)
{
 Console.WriteLine(namegroup.Key);
 foreach (var obj in namegroup.peos)
 Console.WriteLine(obj.Name + "\t" + obj.Age + "\t");
}
```

# Gom nhóm bằng hàm ToLookup

```
//hàm ToLookup = gom nhóm theo Country
Console.WriteLine("Hàm ToLookup");
var ToLook = peoples
 .ToLookup(s => s.Country)
 .Select(std => new
 {
 Key = std.Key,
 peos = std.Select(c => new { c.Name, c.Country })
 });

foreach (var namegroup in ToLook)
{
 Console.WriteLine(namegroup.Key);
 foreach (var obj in namegroup.peos)
 Console.WriteLine(obj.Name + "\t" + obj.Country + "\t");
}
```

# Cú pháp truy vấn Query Syntax – phép kết

```
public class Countries
{
 private string country { get; set; }
 private string countryName { get; set; }
 public string Country
 {
 get { return country; }
 set { country = value; }
 }
 public string CountryName
 {
 get { return countryName; }
 set { countryName = value; }
 }
 public Countries(){}
}
```



# Cú pháp truy vấn Query Syntax – phép kết

## ❖ Kết hợp (join) tương tự **phép kết** trong SQL.

```
var peoples = new List<People>
{
 new People{Name="Tung",Age = 20, Country ="Viet Nam"},
 new People{Name="Trung",Age = 19, Country ="Viet Nam"},
 new People{Name="Chi",Age = 21, Country ="Viet Nam"},
 new People{Name="Tuan",Age = 18, Country ="US"},
 new People{Name="Trinh",Age = 21, Country ="EN"},
 new People{Name="Nha",Age = 24, Country ="UK"},
};

var countries = new List<Countries>
{
 new Countries {Country ="Viet Nam", CountryName = "Việt Nam" },
 new Countries {Country ="US", CountryName = "Mỹ" },
 new Countries {Country ="EN", CountryName = "Anh Quốc" }
};
```

# Cú pháp truy vấn Query Syntax – phép kết

```
var jj = from p in peoples
 join c in countries on p.Country equals c.Country
 select new
 {
 Name = p.Name,
 CountryName = c.CountryName
 };

foreach (var obj in jj)
{
 Console.WriteLine(obj.Name + "\t" + obj.CountryName);
}
```

# Cú pháp truy vấn Query Syntax – phép kết

|       |          |
|-------|----------|
| Tung  | Việt Nam |
| Trung | Việt Nam |
| Chi   | Việt Nam |
| Tuan  | Mỹ       |
| Trinh | Anh Quốc |

# Các phép toán Hội, Trừ, Giao – Dữ liệu

```
public class Countries
{
 private string country { get; set; }
 private string countryName { get; set; }
 public string Country
 {
 get { return country; }
 set { country = value; }
 }
 public string CountryName
 {
 get { return countryName; }
 set { countryName = value; }
 }
}
```

# Các phép toán Hội, Trừ, Giao – Dữ liệu

```
var countries = new List<Countries>
{
 new Countries {Country ="Viet Nam", CountryName = "Việt Nam" },
 new Countries {Country ="US", CountryName = "Mỹ" },
 new Countries {Country ="EN", CountryName = "Anh Quốc" }
};

var countries1 = new List<Countries>
{
 new Countries {Country ="US", CountryName = "Mỹ"},
 new Countries {Country ="EN", CountryName = "Anh Quốc"},
 new Countries {Country ="UK", CountryName = "Vương Quốc Anh"}
};
```

# Cú pháp truy vấn Method Syntax – Union

```
Console.WriteLine("Hội 2 tập hợp");
var uni = countries
 .Select(x => x.Country)
 .Union(countries1
 .Select(y => y.Country)).ToList();
foreach (var obj in uni)
{
 Console.WriteLine(obj);
}
```

# Cú pháp truy vấn Method Syntax – Union

- Hội trên nhiều thuộc tính

```
var uni = countries
 .Select(x => new { x.Country, x.CountryName })
 .Union(countries1
 .Select(x => new { x.Country, x.CountryName })).ToList();

foreach (var obj in uni)
{
 Console.WriteLine(obj.Country + "\t" + obj.CountryName);
}
```

# Cú pháp truy vấn Method Syntax – Except

```
//phép trừ 2 tập hợp
Console.WriteLine("Trừ 2 tập hợp");
var ex = countries
 .Select(c => new {c.Country, c.CountryName})
 .Except(countries1
 .Select(c1 => new {c1.Country, c1.CountryName}));
foreach (var obj in ex)
{
 Console.WriteLine(obj.Country + "\t" + obj.CountryName);
}
```



# Cú pháp truy vấn Method Syntax – Intersect

```
//giao hai tập hợp
Console.WriteLine("Giao 2 tập hợp");
var inter = countries
 .Select(c1 => new { c1.Country, c1.CountryName })
 .Intersect(countries1
 .Select(c2 => new { c2.Country, c2.CountryName}));
foreach (var obj in inter)
{
 Console.WriteLine(obj.Country + "\t" + obj.CountryName);
}
```

# Bài tập

- Cài đặt lớp sinh viên: Tên sinh viên, mã số sinh viên, điểm trung bình, giới tính
  - + Properties: get, set
- Main:
  - + Tạo một danh sách có 5 sinh viên
  - + Sử dụng cú pháp method trong LINQ để thực thi các yêu cầu sau:
  - + In danh sách mới tạo
  - + Liệt kê sinh viên có điểm trung bình  $\geq 5$
  - + Tìm sinh viên có tên = x (x nhập)
  - + Sắp xếp danh sách sinh viên tang theo DTB
  - + Gom nhóm sinh viên theo giới tính, sau đó sắp xếp tăng theo điểm trung bình
  - + Cho biết điểm trung bình cao nhất trong từng giới tính



# Entity Framework

# Giới thiệu

- ❖ Entity Framework (EF) là một framework ánh xạ quan hệ đối tượng (ORM) dành cho ADO.NET
- ❖ Lợi ích lớn nhất của EF là giúp lập trình viên giảm thiểu việc lập trình mã nguồn cần thiết để truy cập và tương tác với cơ sở dữ liệu
- ❖ Được Microsoft phát triển từ năm 2008
- ❖ EF được Microsoft hỗ trợ phát triển lâu dài và bền vững, vì vậy EF là 1 framework mạnh nhất hiện nay để phát triển ứng dụng Web với sự hỗ trợ đồng đẳng của các nhà phát triển Web
- ❖ CSDL được thể hiện bằng 1 lớp con của lớp DbContext
- ❖ Mỗi bảng trong CSDL được đại diện bởi một object của DbSet<T>
- ❖ Mỗi dòng dữ liệu của bảng được thể hiện bằng một Object của lớp tương ứng của bảng
- ❖ Mỗi cột được thể hiện bằng 1 thuộc tính của lớp

# Kết nối CSDL và ánh xạ lớp với bảng dữ liệu

```
public class ProductsContext: DbContext
{
 private const string connectionString
="server=localhost;port=3306;database=myproduct;uid=root;password=";
 protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
 {
 base.OnConfiguring(optionsBuilder);
 optionsBuilder.UseMySQL(connectionString);
 }
 public DbSet<Product> Product { set; get; } // Bảng Product trong DataBase, <Product> tên lớp
}
```

# Bảng dữ liệu

---

| ProductId | Name       | Provider           |
|-----------|------------|--------------------|
| 1         | Iphone     | Xuân Hùng          |
| 2         | Iphone 20  | Công ty 1          |
| 3         | Sản phẩm 2 | Công ty 1          |
| 9         | Iphone 20  | Công ty Phương Nam |

# Xây dựng lớp tương ứng

```
public class Product
{
 private int productid;
 private string name;
 private string provider;
 public int ProductId { set; get; }
 public string Name { set; get; }
 public string Provider { set; get; }
}
```

# Truy xuất danh sách sản phẩm

```
public static void ReadProducts()
{
 using (var context = new ProductsContext())
 {
 // Lấy List các sản phẩm từ DB
 var products = context.Product.ToList();
 Console.WriteLine("Tất cả sản phẩm");
 foreach (var product in products)
 {
 Console.WriteLine($"{product.ProductId,2} {product.Name,10} - {product.Provider}");
 }
 }
}
```



# Truy xuất danh sách sản phẩm – có điều kiện

```
public static void ReadProducts()
{
 using (var context = new ProductsContext())
 {
 var products = context.Product;
 var sanpham = products
 .Where(p => p.ProductId == 1);

 foreach (var product in sanpham)
 {
 Console.WriteLine($"{product.ProductId,2} {product.Name,10} - {product.Provider}");
 }
 }
}
```

# Thêm sản phẩm

```
public static void InsertProduct()
{
 using (var context = new ProductsContext())
 {
 // Dùng context để thêm
 int id = 9;
 string name = "iphone 6s plus";
 string pro = "Công ty Phương Nam";
 context.Product.Add(new Product()
 {
 ProductId = id,
 Name = name,
 Provider = pro
 });
 // Thực hiện Insert vào DB các dữ liệu đã thêm.
 int rows = context.SaveChanges();
 Console.WriteLine($"Đã lưu {rows} sản phẩm");
 }
}
```

# Cập nhật sản phẩm

```
public static void RenameProduct(int id, string newName)
{
 using (var context = new ProductsContext())
 {
 //dung cú pháp method LINQ
 var product = context.Product
 .Where(d => d.ProductId == id)
 .FirstOrDefault();
 if (product != null)
 {
 product.Name = newName;
 //product.Provider = Provider;
 Console.WriteLine($"{product.ProductId,2} có tên mới = {product.Name,10}");
 context.SaveChanges(); //Thi hành cập nhật
 }
 }
}
```

# Xóa sản phẩm

```
public static void DeleteProduct(int id)
{
 using (var context = new ProductsContext())
 {
 var product = (from p in context.Product
 where (p.ProductId == id)
 select p
)
 .FirstOrDefault(); // Lấy Product có ID chỉ ra

 if (product != null)
 {
 context.Remove(product);
 Console.WriteLine($"Xóa {product.ProductId}");
 context.SaveChanges();
 }
 }
}
```