



# Chương 4: Kết nối CSDL

**Giảng viên:** Vũ Minh Sang  
**Phòng:** E9.4  
**E-mail:** sangvm@uit.edu.vn

# Nội dung



**ADO.NET**



**LINQ (Language-Integrated Query)**



# ADO.NET

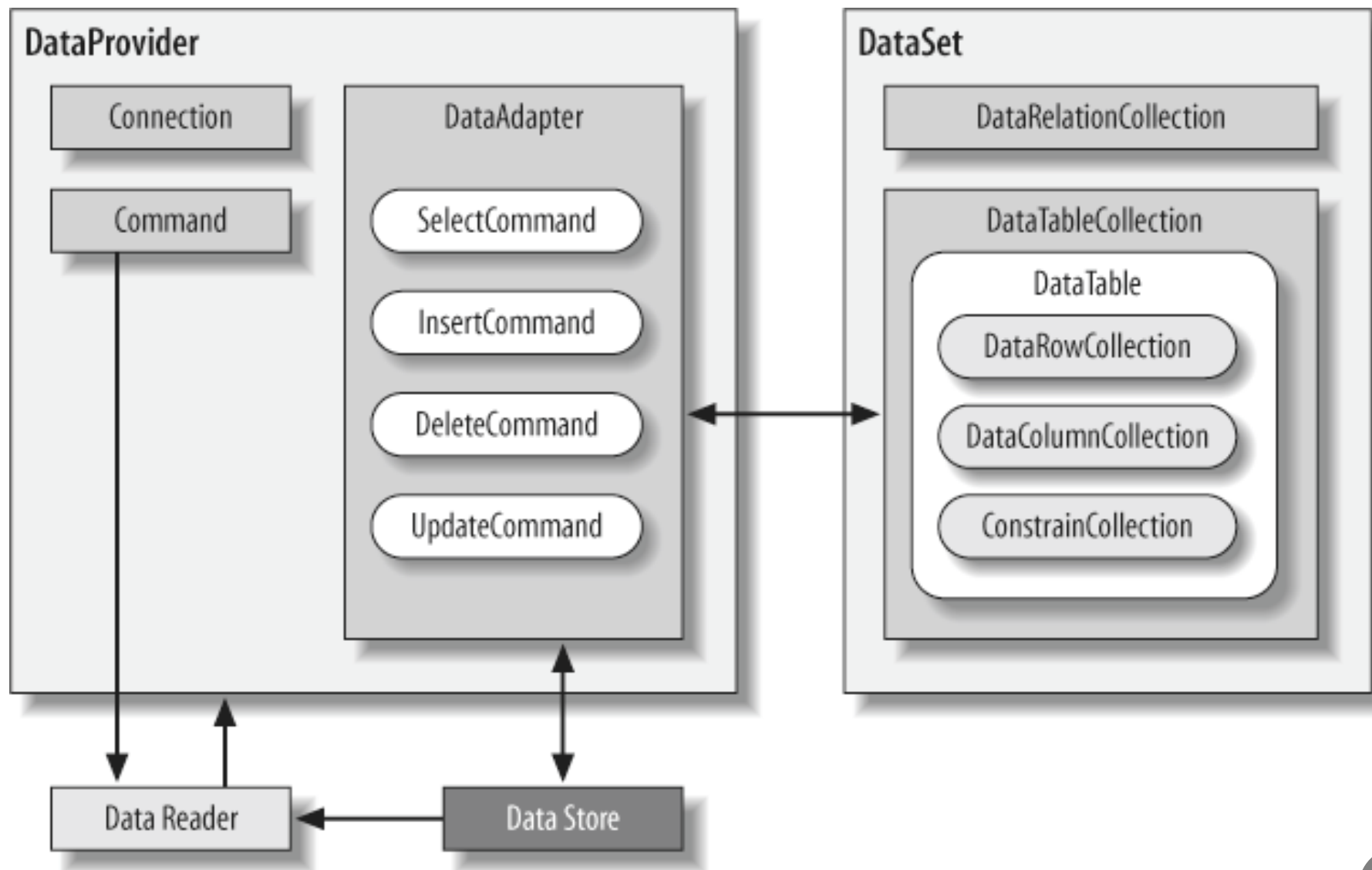
# ADO.NET

- 📖 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.

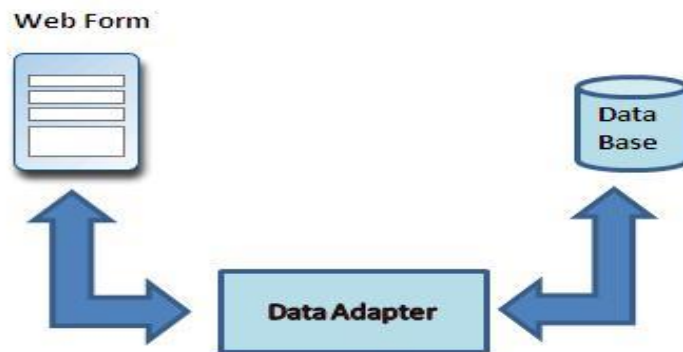


# Thành phần Connected

- 📖 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.
  - 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 sao này xuống lại CSDL



Disconnected Architecture

# 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.

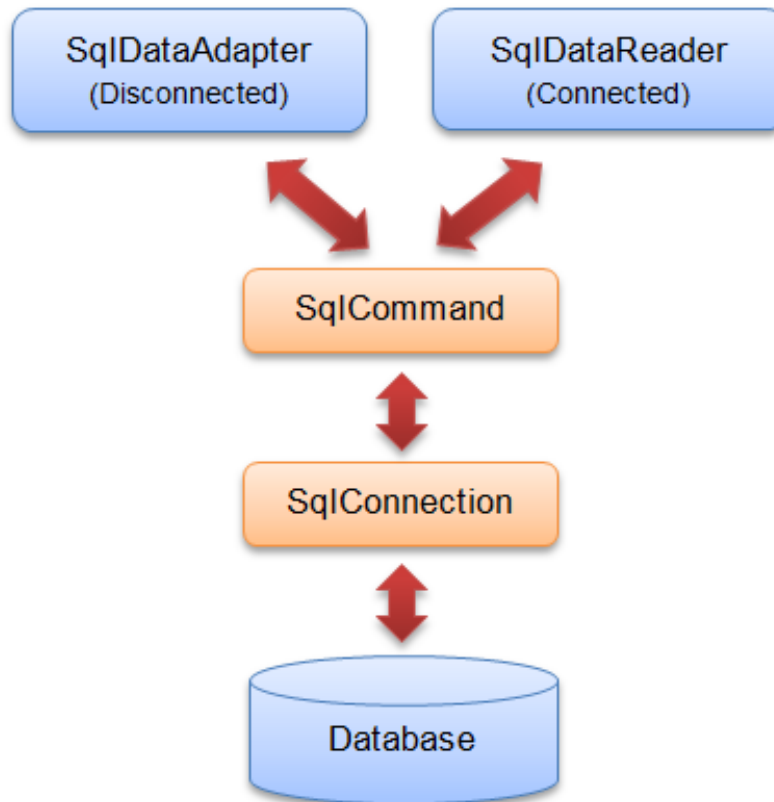
📖 VD: sử dụng cho SQL Server:

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

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

# Kết nối CSDL với Connected

- 📖 B1: Tạo thông số kết nối và kết nối CSDL (`SqlConnection`)
- 📖 B2: Tạo câu lệnh SQL và thực thi (`SqlCommand`)
- 📖 B3: Xử lý kết quả trả về (`SqlDataReader`)
- 📖 B4: Đóng kết nối



# 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**

Chuỗi kết nối đầy đủ:

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

# Thông số kết nối CSDL

- 📖 Sử dụng lớp `ConnectionStringBuilder` tạo chuỗi kết nối.

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

- 📖 Gán vào chuỗi kết nối

```
📖 var conString = conStringBuilder.ToString();
```

# 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.

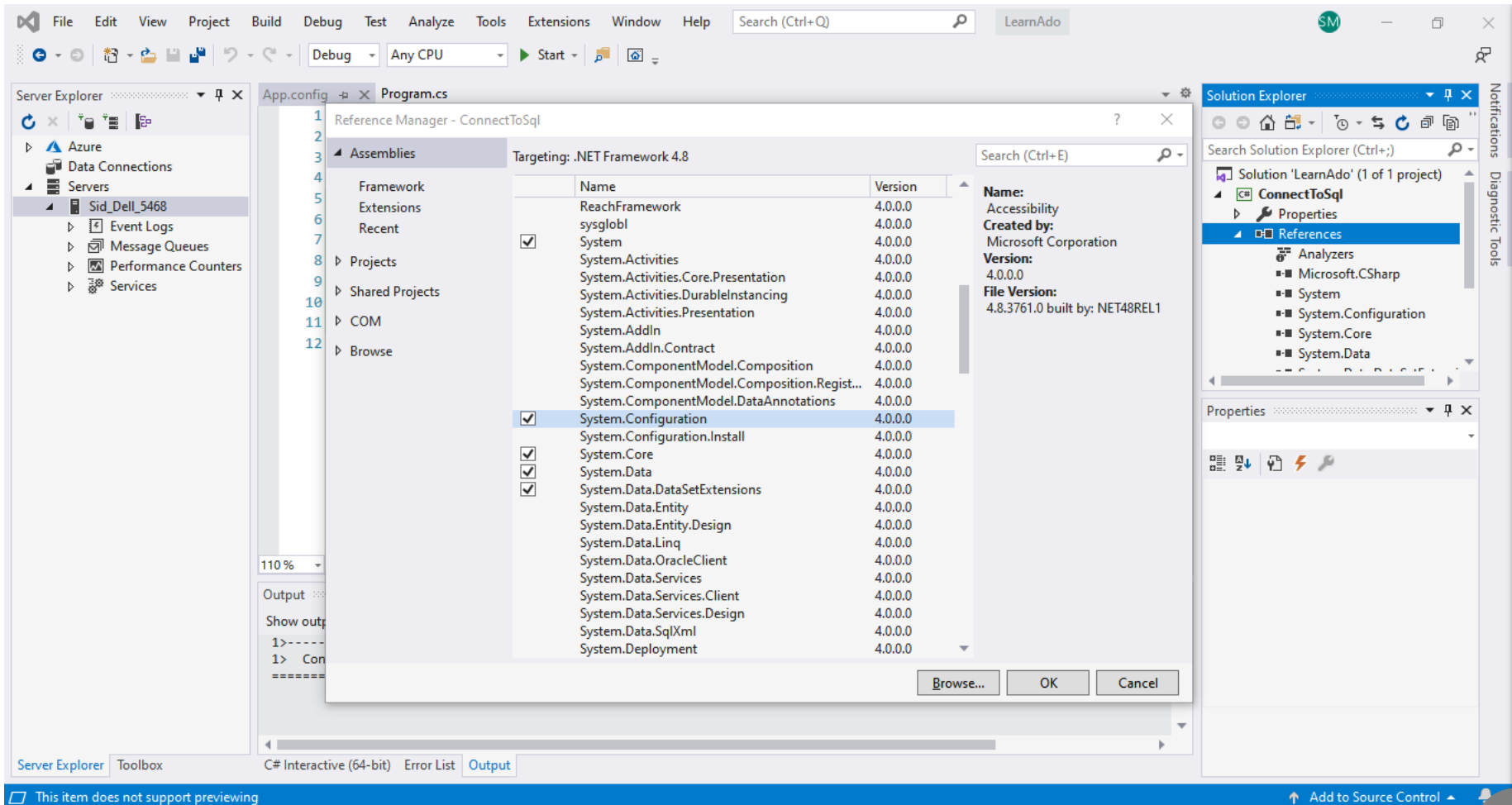
```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <startup>
    <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.8" />
  </startup>
</configuration>
```

- 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

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

○ Cách 1:

```
var connection = new SqlConnection();  
connection.ConnectionString = conString;
```

○ Cách 2:

```
var connection = new SqlConnection(conString);
```

---

○ Cách 3:

```
var connection = new SqlConnection  
{  
    ConnectionString = conString  
};
```

---

# 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();
```

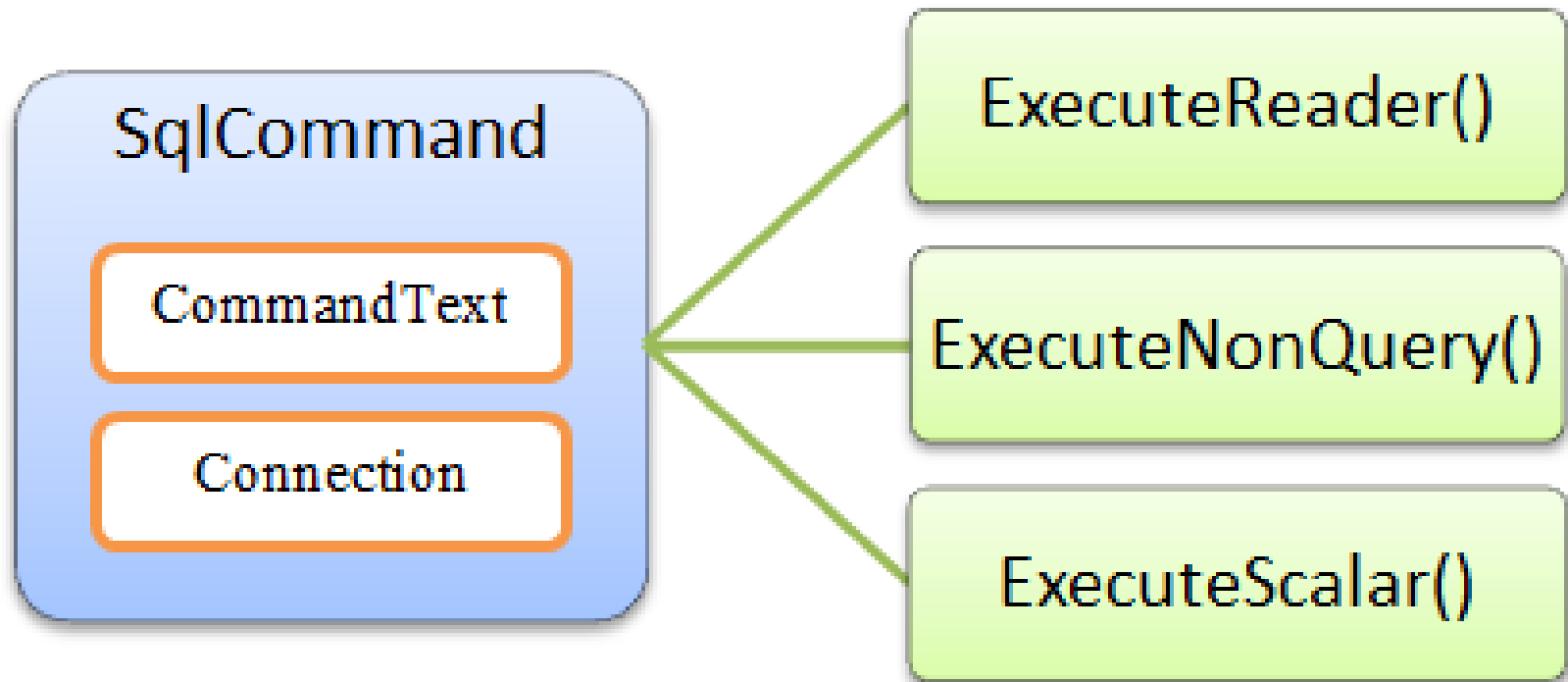
```
var queryString = "Select * from Test";
```

```
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

# Tạo câu lệnh SQL và thực thi





# 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.

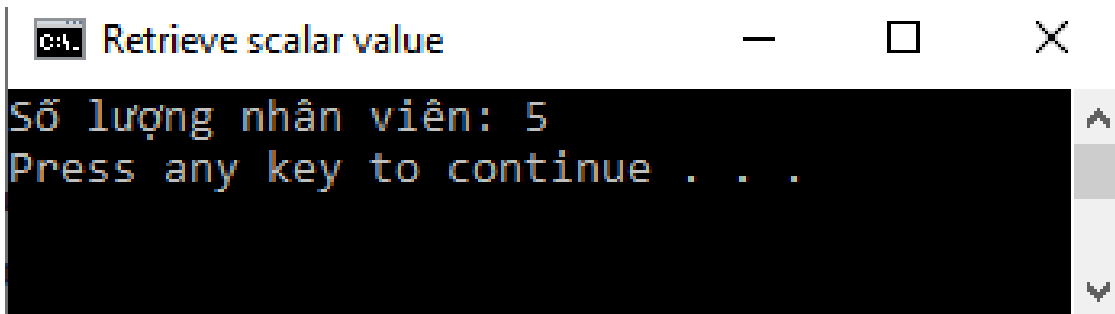
📖 Các giá trị của **CommandBehavior**:

- **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)
- 📖 Giá trị trả về là object, cần ép kiểu để sử dụng.

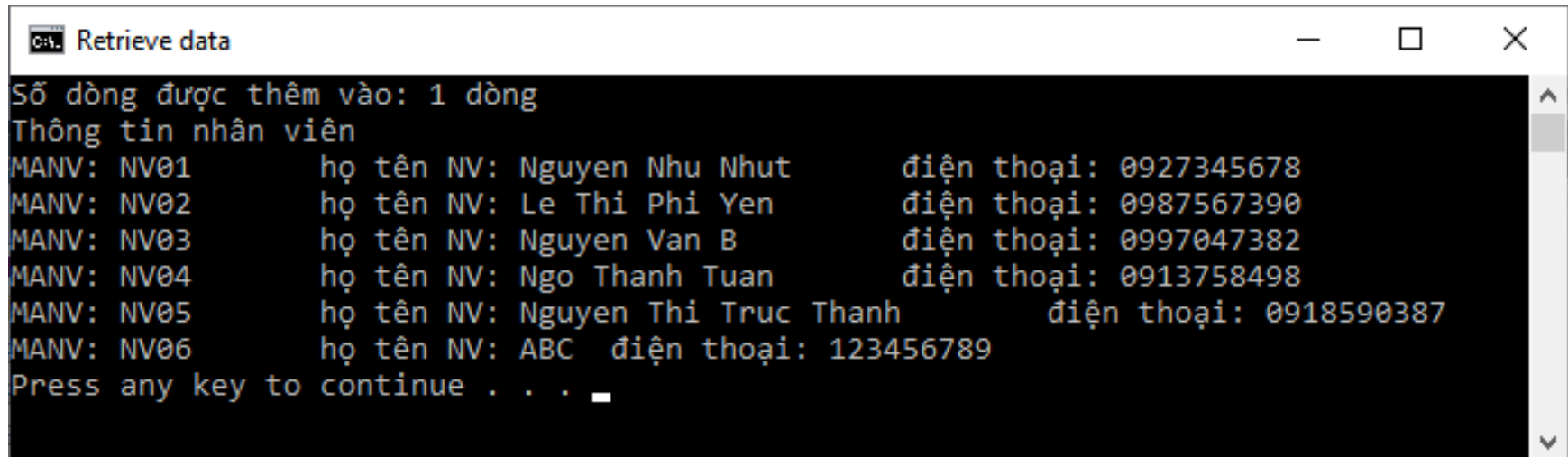
```
Console.Title = "Retrieve scalar value";
Console.OutputEncoding = Encoding.UTF8;
var conString = @"Data Source=localhost;Initial Catalog=QLBH;Integrated Security=True";
using (var connection = new SqlConnection(conString))
{
    var queryString = "Select count(*) from NHANVIEN";
    var command =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



```
Retrieve scalar value
Số lượng nhân viên: 5
Press any key to continue . . .
```

# 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

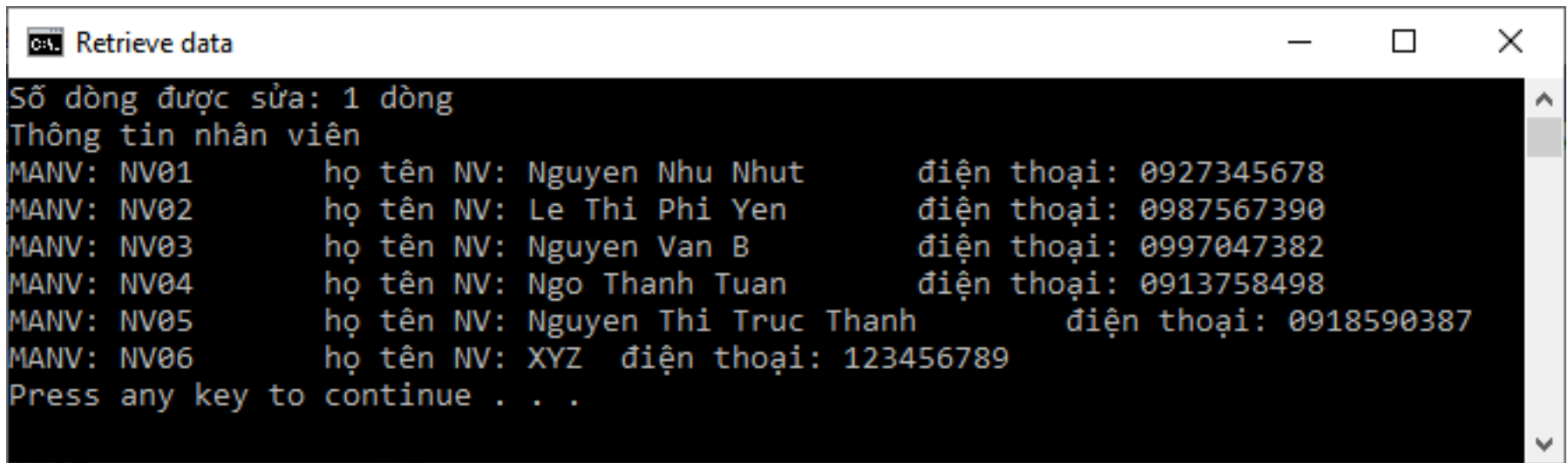
Thông tin nhân viên

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

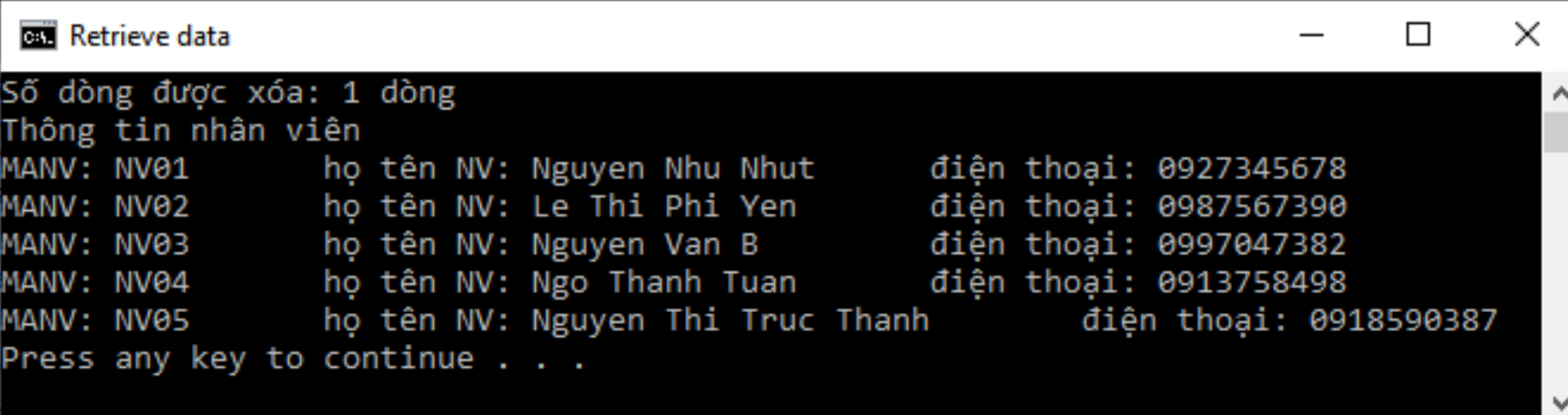
```
var conString = @"Data Source=localhost;Initial Catalog=QLBH;Integrated Security=True";
using (var connection = new SqlConnection(conString))
using (var command = new SqlCommand { Connection = connection })
{
    var updateString = "Update NHANVIEN set HOTEN = 'XYZ' where MANV = 'NV06'";
    command.CommandText = 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");
}
```



The screenshot shows a console window with the title "Retrieve data". The output text is as follows:

```
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);
```

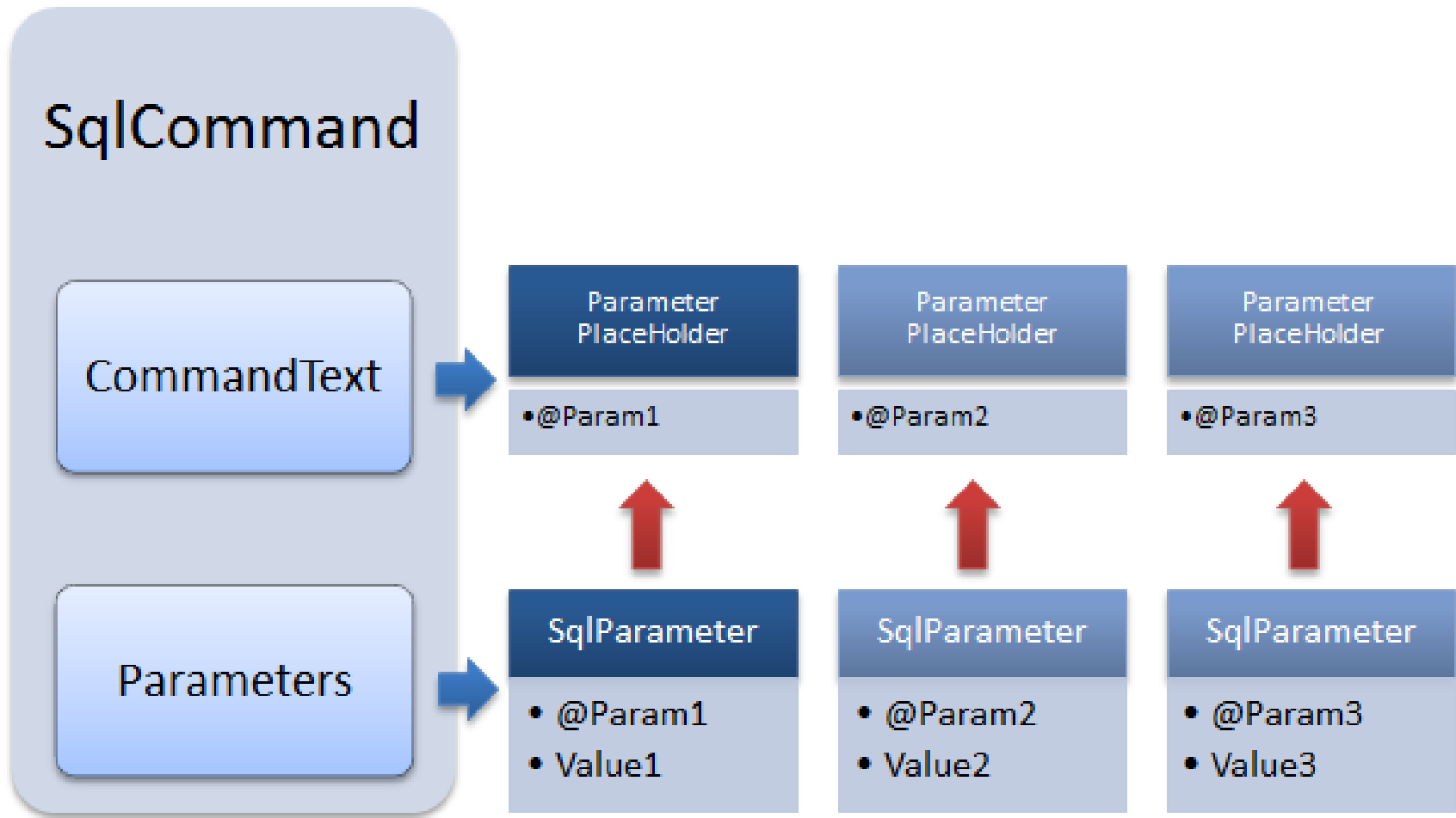
📖 Gán object `SqlParameter` vào thuộc tính `Parameters` của `SqlCommand`.

```
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



# 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

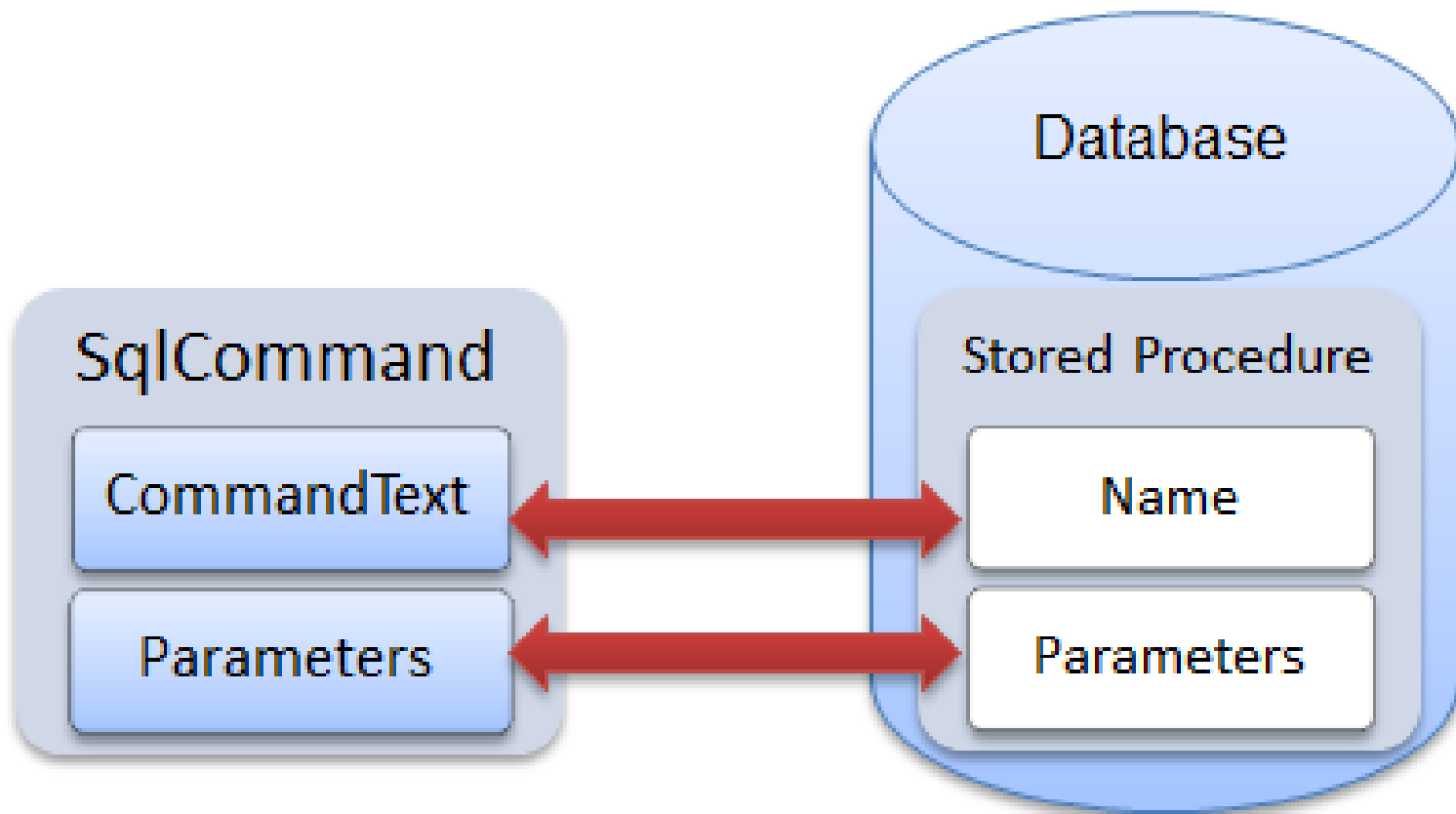
```
Console.Write("Thêm mã nhân viên: ");
var ma = Console.ReadLine();
Console.Write("Thêm họ tên nhân viên: ");
var hoTen = Console.ReadLine();
Console.Write("Thêm số điện thoại: ");
var dt = Console.ReadLine();
Console.Write("Thêm ngày sinh: ");
var ns = Console.ReadLine();

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(conString))
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.WriteLine("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}");
        }
    }
}
```





# LINQ (Language-Integrated Query)

# 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 .
- 📖 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`.

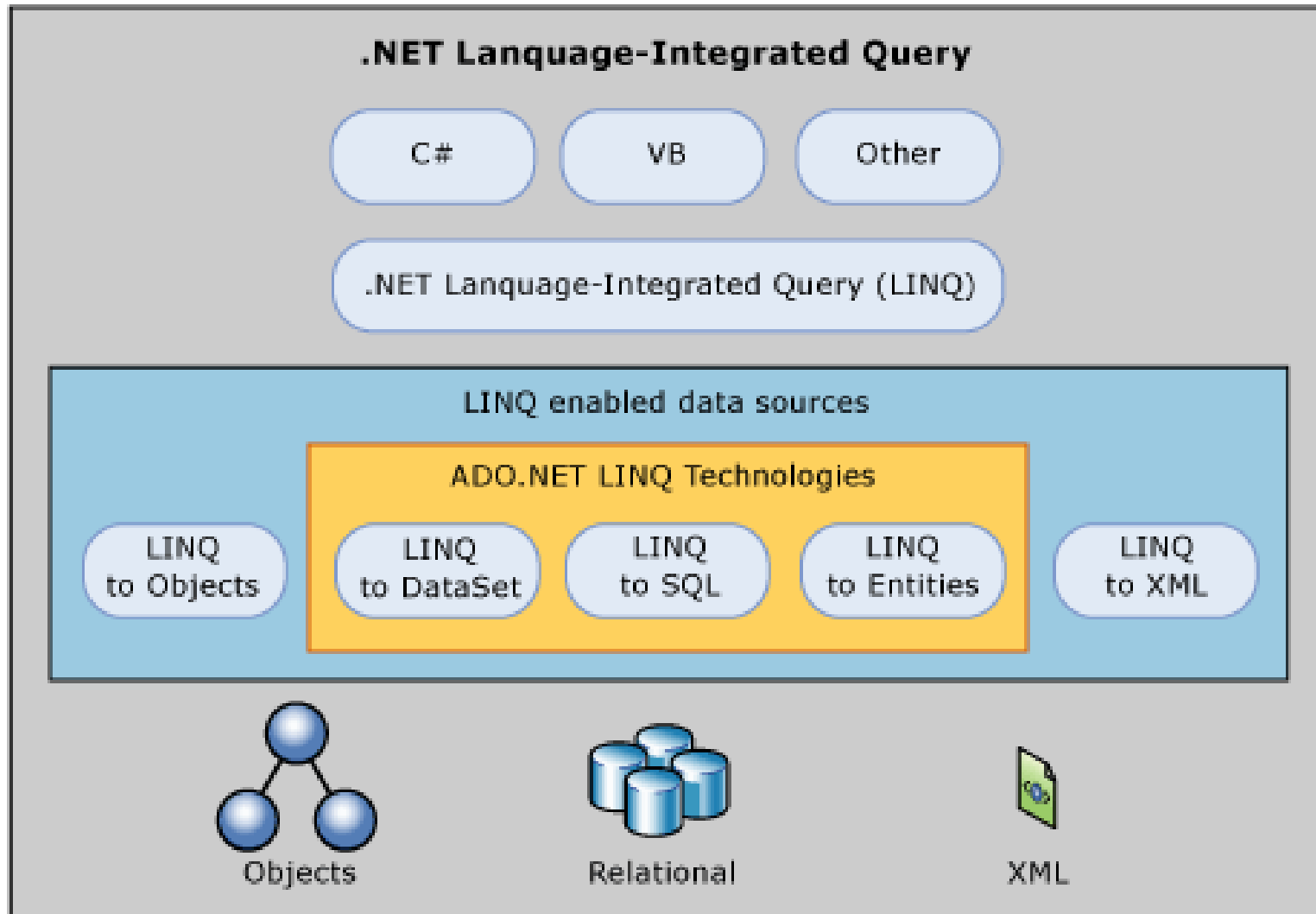
Biểu thức truy vấn tương tự trong SQL với các toán tử truy vấn: From, Where... và thường bắt đầu bằng From.

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.

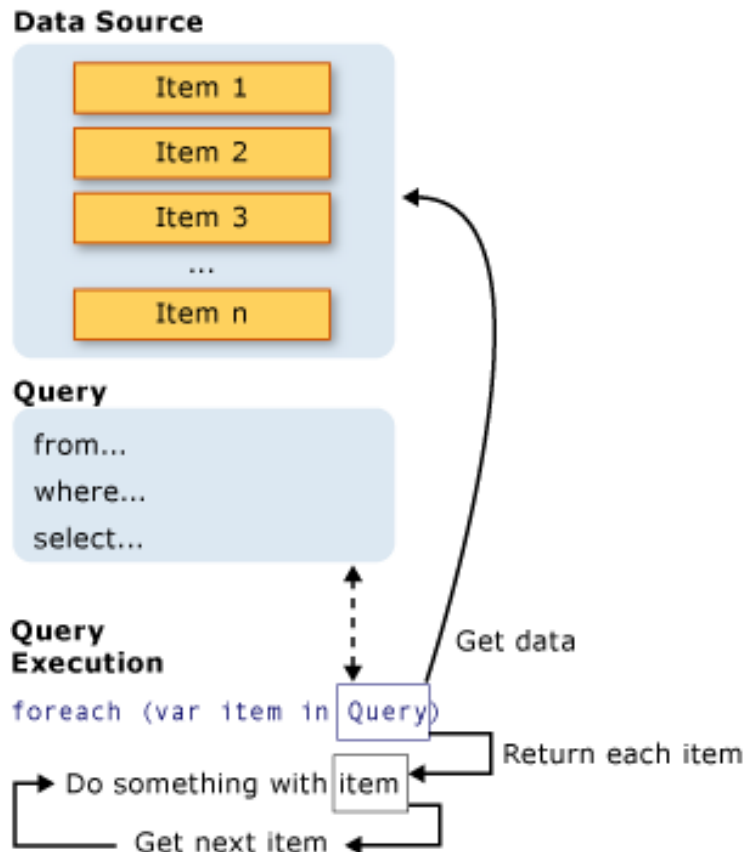
# Kiến trúc của LINQ



# 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, OrderByDescending, 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

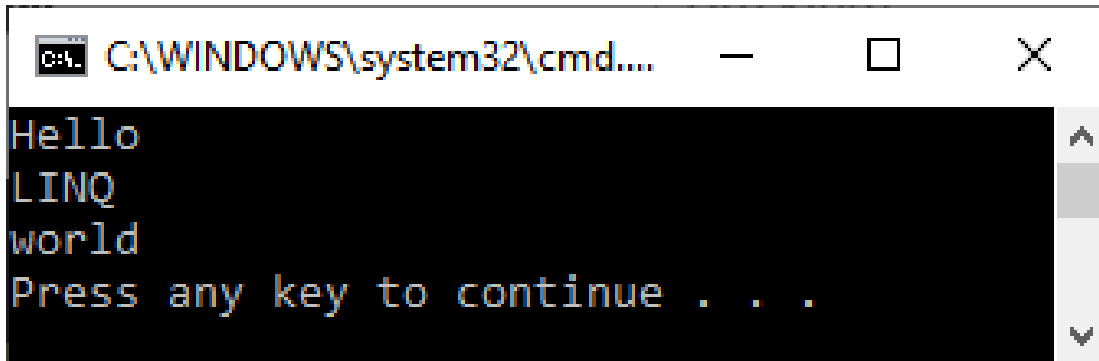
Thực thi truy vấn:

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
string[] words = { "Hello", "LINQ", "world" };
//Tạo truy vấn
var shortWords = from word in words where word.Length <= 5 select word;
//Thực thi truy vấn
foreach (var word in shortWords)
{
    Console.WriteLine(word);
}
```

A screenshot of a Windows Command Prompt window. The title bar shows the path 'C:\WINDOWS\system32\cmd....'. The command prompt displays the output of the LINQ query: 'Hello', 'LINQ', and 'world', each on a new line. Below the output, it says 'Press any key to continue . . .'. The window has standard Windows window controls (minimize, maximize, close) in the title bar.

```
C:\WINDOWS\system32\cmd....
Hello
LINQ
world
Press any key to continue . . .
```

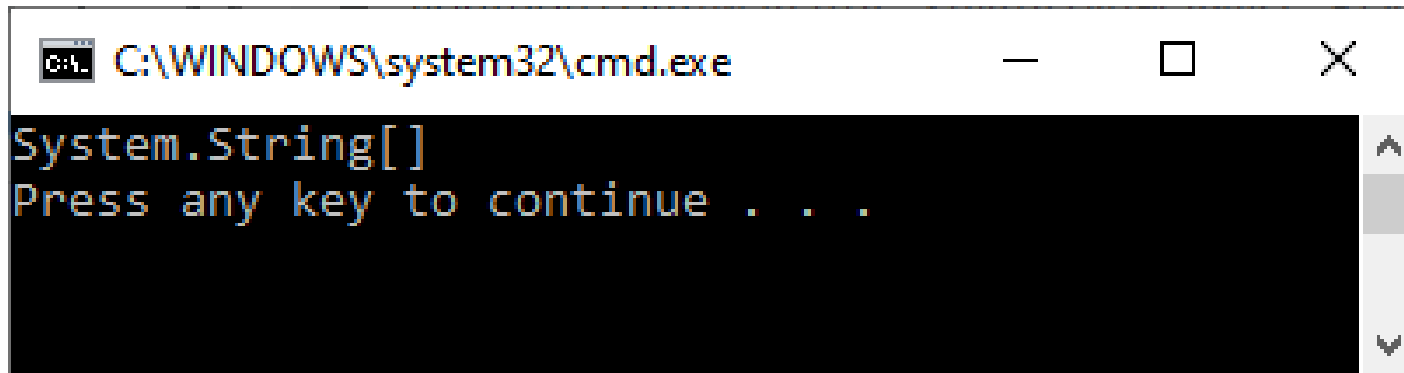
# Truy vấn trong LINQ

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

```
//Tạo nguồn dữ liệu
string[] words = { "Hello", "LINQ", "world" };
//Tạo truy vấn
var shortWords = (from word in words where word.Length <= 5 select word).ToArray();
//Thực thi truy vấn
Console.WriteLine(shortWords);
```

A screenshot of a Windows command prompt window. The title bar shows the path 'C:\WINDOWS\system32\cmd.exe'. The command prompt displays the output of the LINQ query: 'System.String[]' on the first line and 'Press any key to continue . . .' on the second line. The text is in a monospaced font, with 'System.String[]' in a light blue color and the rest in white. There are up and down arrow icons on the right side of the command prompt area.

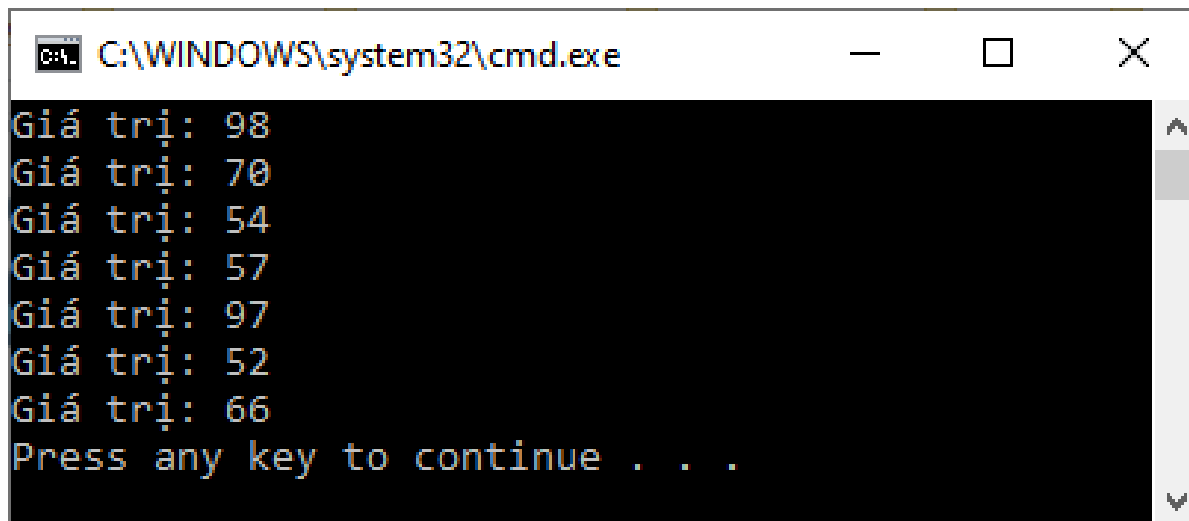
```
C:\WINDOWS\system32\cmd.exe
System.String[]
Press any key to continue . . .
```

# Biểu thức Lambda

- 📖 Biểu thức Lambda (Lambda Expression): làm một hàm ẩn 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

```
//Biểu thức Lambda
int[] mang = new[] { 6, 98, 12, 34, 70, 54, 57, 97, 52, 11, 66 };
foreach (int i in mang.Where(x =>
{
    if (x <= 10) return false;
    else if (x >= 50) return true;
    return false;
}))
    Console.WriteLine("Giá trị: {0}", i);
```



```
C:\WINDOWS\system32\cmd.exe
Giá trị: 98
Giá trị: 70
Giá trị: 54
Giá trị: 57
Giá trị: 97
Giá trị: 52
Giá trị: 66
Press any key to continue . . .
```



# VD truy vấn trong LINQ

📖 Xét tập dữ liệu là một danh sách

```
var dinosaurs = new List<Dinosaurs>
{
    new Dinosaurs { Name = "Tyrannosaurus", Age = 10, Country = "US" },
    new Dinosaurs { Name = "Amargasaurus", Age = 20, Country = "AU" },
    new Dinosaurs { Name = "Deinonychus", Age = 8, Country = "UK" },
    new Dinosaurs { Name = "Acrocanthosaurus", Age = 15, Country = "US" },
    new Dinosaurs { Name = "Albertosaurus", Age = 5, Country = "UK" },
    new Dinosaurs { Name = "Carnotaurus", Age = 25, Country = "EU" },
    new Dinosaurs { Name = "Baryonyx", Age = 26, Country = "UK" },
    new Dinosaurs { Name = "Compsognathus", Age = 111, Country = "EU" },
    new Dinosaurs { Name = "Daspletosaurus", Age = 16, Country = "US" },
    new Dinosaurs { Name = "Edmontosaurus", Age = 17, Country = "EU" }
};
```

# 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**>**.

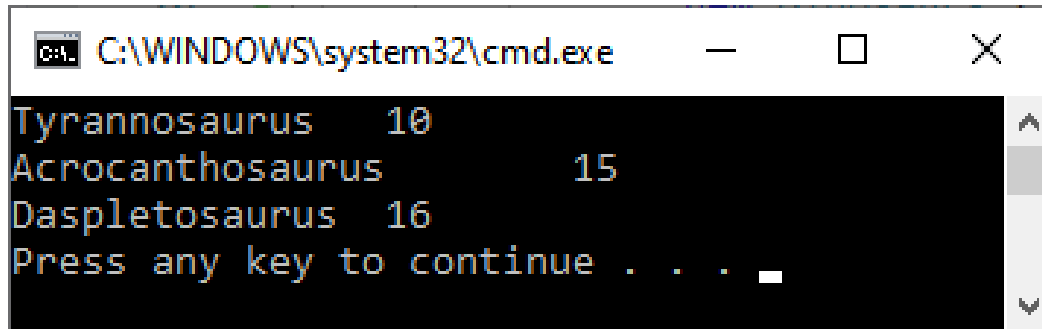
Trong đó **Tsource** là kiểu dữ liệu cơ sở của nguồn dữ liệu.  
Kết quả dữ liệu trả về khi thỏa **true** và sẽ vào danh sách và truy xuất như mảng

# Một số phương thức trong LINQ

- VD: Liệt kê danh sách Dinosaurs ở US.
- Vị trí tham số có thể cung cấp bằng hàm lambda hoặc các loại phương thức.

//Sử dụng phương thức where

```
var dino = dinosaurs.Where(d => d.Country == "US");  
foreach (var d in dino) Console.WriteLine($"{d.Name}\t{d.Age}");
```



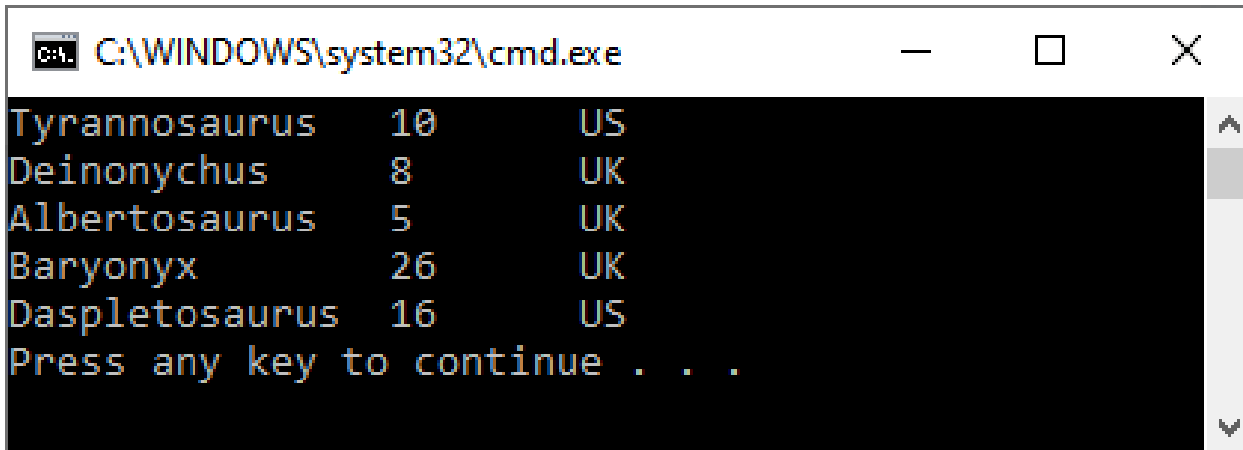
The screenshot shows a Windows command prompt window titled "C:\WINDOWS\system32\cmd.exe". The output of the LINQ query is displayed as follows:

```
Tyrannosaurus 10  
Acrocanthosaurus 15  
Daspletosaurus 16  
Press any key to continue . . .
```

# Một số phương thức trong LINQ

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

```
//Liệt kê danh sách Dinosaurs ở vị trí chẵn  
var even = dinosaurs.Where((d, i) => i % 2 == 0);  
foreach (var d in even) Console.WriteLine($"{d.Name}\t{d.Age}" +  
                                             $"{d.Country}");
```



```
C:\WINDOWS\system32\cmd.exe  
Tyrannosaurus    10      US  
Deinonychus      8       UK  
Albertosaurus    5       UK  
Baryonyx         26      UK  
Daspletosaurus   16      US  
Press any key to continue . . .
```

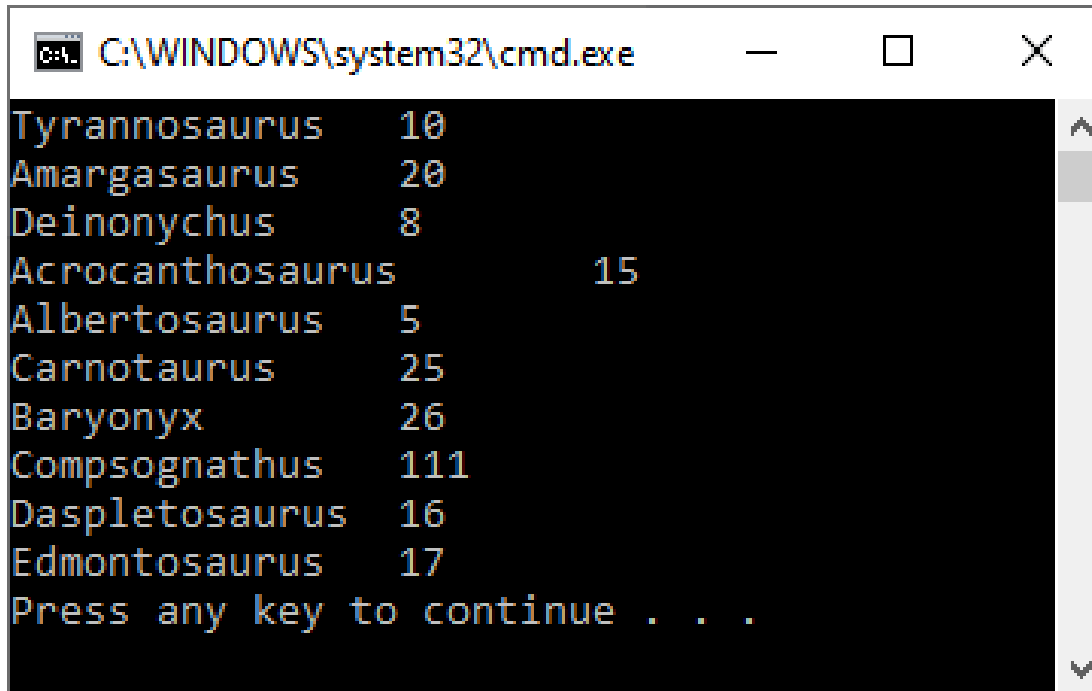
# Một số phương thức trong LINQ

- 📖 Phương thức **Select**: tương tự với truy vấn Select trong SQL.
- 📖 Thực hiện chuyển đổi dữ liệu từ nguồn dữ liệu từ hình thức này sang hình thức khác.
- 📖 VD: Bảng SinhVien lưu trữ thông tin sinh viên: Mã, tên, ngày sinh, email...; nhưng có thể chỉ cần xem một 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 Age của Dinosaurs.
- Kết quả trả về là một chuỗi.

```
//Liệt kê danh sách Name và Age của Dinosaurs  
var info = dinosaurs.Select(d => $"{d.Name}\t{d.Age}");  
foreach (var d in info) Console.WriteLine(d);
```



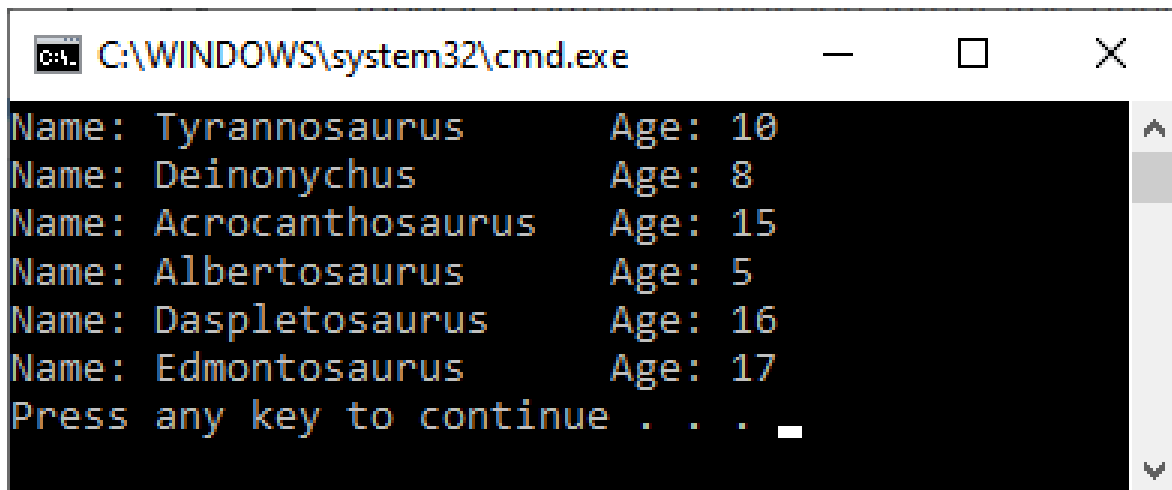
```
C:\WINDOWS\system32\cmd.exe  
Tyrannosaurus    10  
Amargasaurus    20  
Deinonychus      8  
Acrocanthosaurus 15  
Albertosaurus    5  
Carnotaurus      25  
Baryonyx         26  
Compsognathus    111  
Daspletosaurus   16  
Edmontosaurus    17  
Press any key to continue . . .
```

# Một số phương thức trong LINQ

📖 VD: Liệt kê danh sách Name và Age của Dinosaurs với Age < 20

📖 Lọc dữ liệu với Where rồi Select.

```
//Liệt kê danh sách Name và Age của Dinosaurs với Age < 20
var youngDino = dinosaurs
    .Where(d => d.Age < 20)
    .Select(d => new { Info = $"{d.Name}", d.Age });
foreach (var d in youngDino) Console.WriteLine($"Name: {d.Info}\t Age: {d.Age}");
```



```
C:\WINDOWS\system32\cmd.exe
Name: Tyrannosaurus      Age: 10
Name: Deinonychus        Age: 8
Name: Acrocanthosaurus   Age: 15
Name: Albertosaurus      Age: 5
Name: Daspletosaurus     Age: 16
Name: Edmontosaurus      Age: 17
Press any key to continue . . .
```

# Cú pháp truy vấn trong LINQ

📖 VD: Liệt kê toàn bộ danh sách Dinosaurs

```
//Liệt kê toàn bộ danh sách Dinosaurs
var queryAllDino = from dino in dinosaurs
                   select dino;
foreach (var d in queryAllDino) Console.WriteLine($"{ d.Name}\t" +
                                                    $"{ d.Age}\t{d.Country }");
```

📖 VD: Liệt kê danh sách Dinosaurs theo Name

```
//Liệt kê danh sách Dinosaurs theo Name
var queryAllDino = from dino in dinosaurs
                   select dino.Name;
foreach (var d in queryAllDino) Console.WriteLine($"{d}");
```



# Cú pháp truy vấn trong LINQ

Lọc dữ liệu (filter) với mệnh đề Where.

VD: Liệt kê danh sách Dinosaurs ở US.

```
//Liệt kê danh sách Dinosaurs ở US
```

```
var queryDino = from dino in dinosaurs
                 where dino.Country == "US"
                 select dino;
foreach (var d in queryDino) Console.WriteLine($"{d.Name}\t" +
                                                $"{d.Age}\t{d.Country}");
```

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

```
var queryDino = from dino in dinosaurs
                 where dino.Country == "US" && dino.Age > 10
                 select dino;
foreach (var d in queryDino) Console.WriteLine($"{d.Name}\t" +
                                                $"{d.Age}\t{d.Country}");
```

# Cú pháp truy vấn trong LINQ

- 📖 Sắp xếp (order) với mệnh đề Orderby.
- 📖 VD: Sắp xếp danh sách Dinosaurs ở UK theo Name

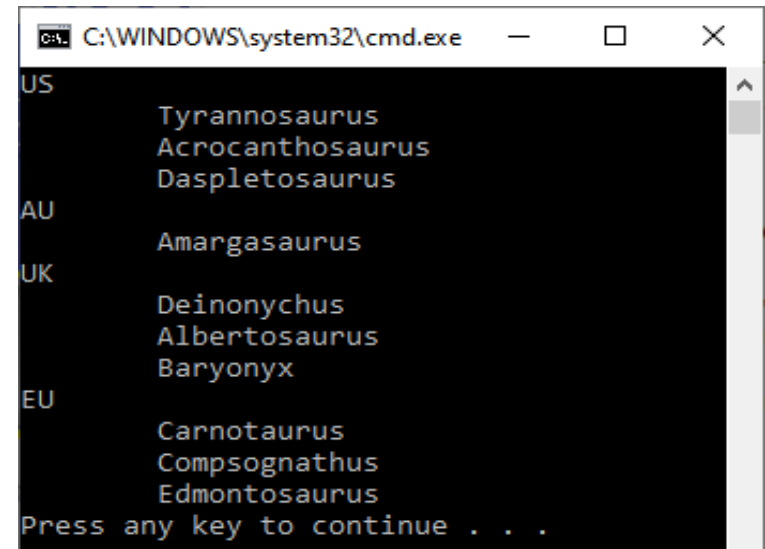
//Sắp xếp theo Name

```
var queryDino = from dino in dinosaurs
                 where dino.Country == "UK"
                 orderby dino.Name ascending
                 select dino;
foreach (var d in queryDino) Console.WriteLine($"{d.Name}\t{d.Age}");
```

# Cú pháp truy vấn trong LINQ

- ☞ 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 Dinosaurs theo Country

```
//Gom nhóm theo Country
var queryDino = from dino in dinosaurs
                group dino by dino.Country;
foreach (var dinoGroup in queryDino)
{
    Console.WriteLine(dinoGroup.Key);
    foreach (Dinosaurs dino in dinoGroup)
        Console.WriteLine($"{dino.Name}");
}
```



```
C:\WINDOWS\system32\cmd.exe
US
    Tyrannosaurus
    Acrocanthosaurus
    Daspletosaurus
AU
    Amargasaurus
UK
    Deinonychus
    Albertosaurus
    Baryonyx
EU
    Carnotaurus
    Compsognathus
    Edmontosaurus
Press any key to continue . . .
```

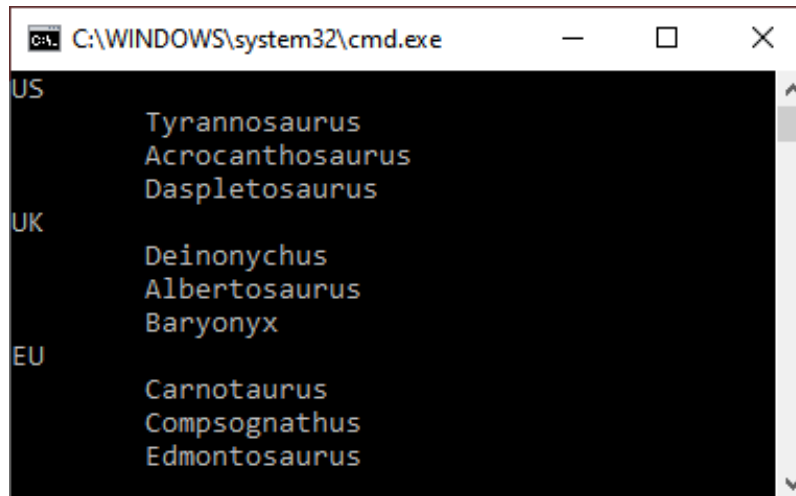
# Cú pháp truy vấn trong LINQ

VD: Gom nhóm Dinosaurs theo Country và số lượng là hơn 2

```
//Gom nhóm hơn 2
```

```
var queryDino = from dino in dinosaurs
                 group dino by dino.Country into dinoGrp
                 where dinoGrp.Count() > 2
                 //orderby dinoGrp.Key
                 select dinoGrp;
```

```
foreach (var dinoGroup in queryDino)
{
    Console.WriteLine(dinoGroup.Key);
    foreach (Dinosaurs dino in dinoGroup)
        Console.WriteLine($"{dino.Name}");
}
```



```
C:\WINDOWS\system32\cmd.exe
US
    Tyrannosaurus
    Acrocanthosaurus
    Daspletosaurus
UK
    Deinonychus
    Albertosaurus
    Baryonyx
EU
    Carnotaurus
    Compsognathus
    Edmontosaurus
```

# Cú pháp truy vấn trong LINQ

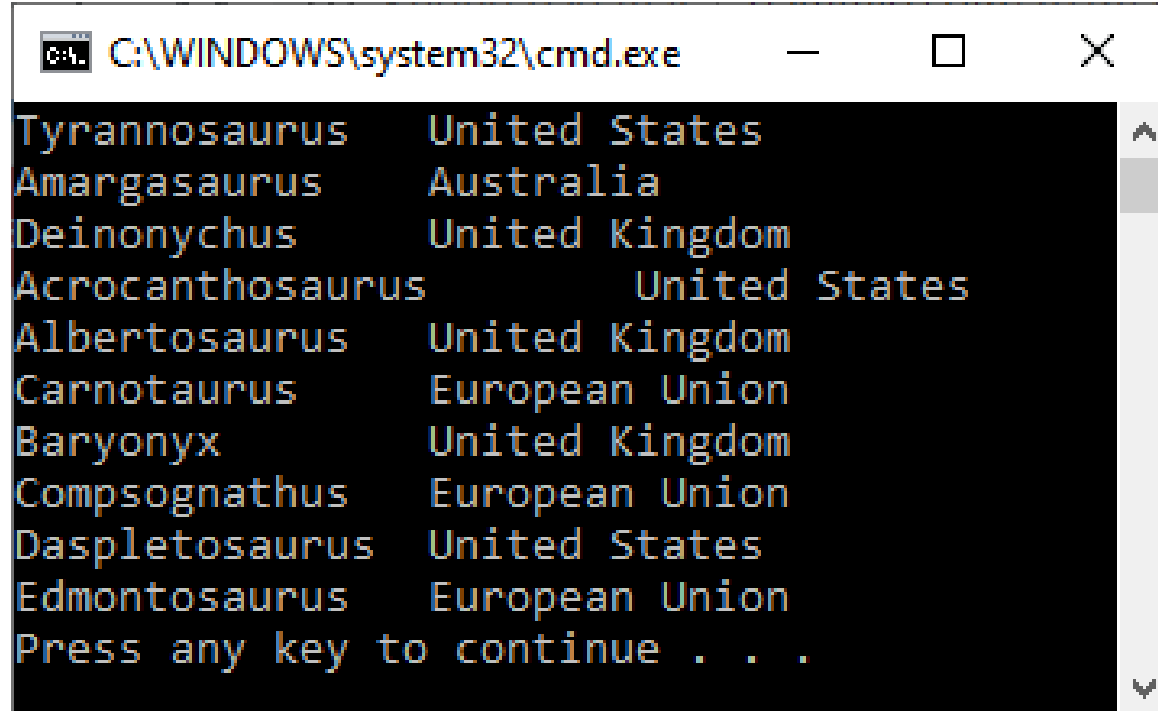
📖 Kết hợp (join) tương tự phép kết trong SQL.

```
var country = new List<Countries>
{
    new Countries {Country = "US", CountryName = "United States"},
    new Countries {Country = "AU", CountryName="Australia"},
    new Countries {Country = "UK", CountryName="United Kingdom"},
    new Countries {Country = "EU", CountryName="European Union"}
};
```

//Kết hợp Join

```
var joinQuery = from dino in dinosaurs
                join co in country on dino.Country equals co.Country
                select new
                {
                    DinoName = dino.Name,
                    CountryName = co.CountryName
                };
foreach (var d in joinQuery) Console.WriteLine($"{d.DinoName}\t{d.CountryName}");
```

# Cú pháp truy vấn trong LINQ



A screenshot of a Windows command prompt window titled "C:\WINDOWS\system32\cmd.exe". The window displays a list of dinosaurs and their locations, with a vertical scrollbar on the right side. The text is as follows:

Tyrannosaurus	United States
Amargasaurus	Australia
Deinonychus	United Kingdom
Acrocanthosaurus	United States
Albertosaurus	United Kingdom
Carnotaurus	European Union
Baryonyx	United Kingdom
Compsognathus	European Union
Daspletosaurus	United States
Edmontosaurus	European Union
Press any key to continue . . .	

# LINQ to SQL

- 📖 LINQ to SQL là phiên bản hiện thực hóa của ORM (Object Relational Mapping).
- 📖 Được dùng để mô hình hóa CSDL trong Sql Server sử dụng các lớp trong .NET.
- 📖 Mỗi bảng trong CSDL sẽ được ánh xạ với một class trong ứng dụng.
- 📖 Cho phép thực hiện các thao tác truy vấn đến CSDL thông qua các class đã được ánh xạ.
- 📖 Hỗ trợ đầy đủ transaction, view và stored procedure.

# LINQ to SQL

 Thêm CSDL vào trong ứng dụng.

- Click phải ở **Data Connections** trong **Server Explorer**; chọn **Add Connection**
- Ở cửa sổ **Add Connection** chọn **Server Name** và **CSDL** muốn add vào ứng dụng.



## Add Connection



Enter information to connect to the selected data source or click "Change" to choose a different data source and/or provider.

Data source:

Microsoft SQL Server (SqlClient)

Change...

Server name:

SID\_DELL\_5468

Refresh

Log on to the server

Authentication: Windows Authentication

User name:

Password:

☐ Save my password

Connect to a database

☒ Select or enter a database name:

Northwind

☐ Attach a database file:

Browse...

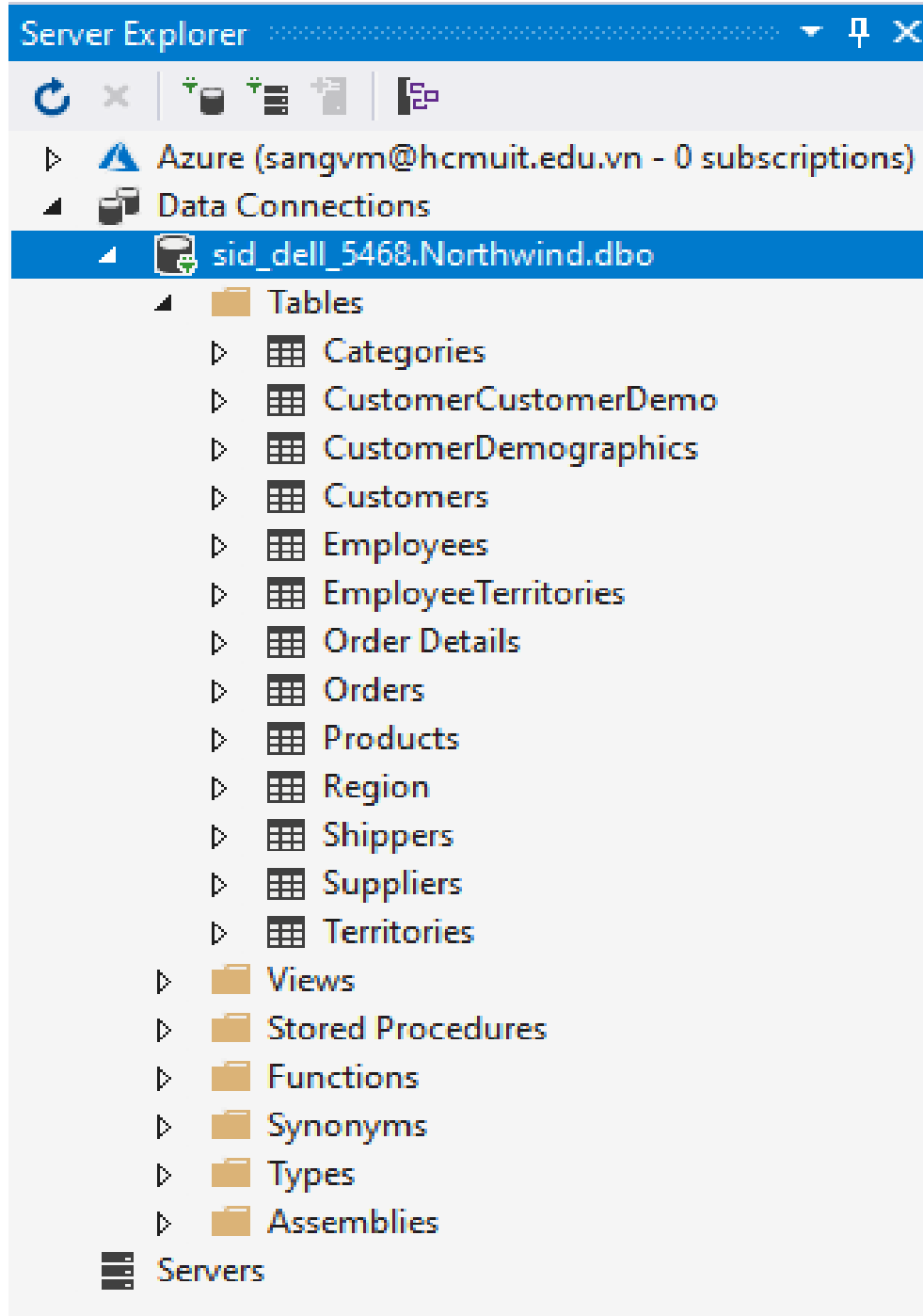
Logical name:

Advanced...

Test Connection

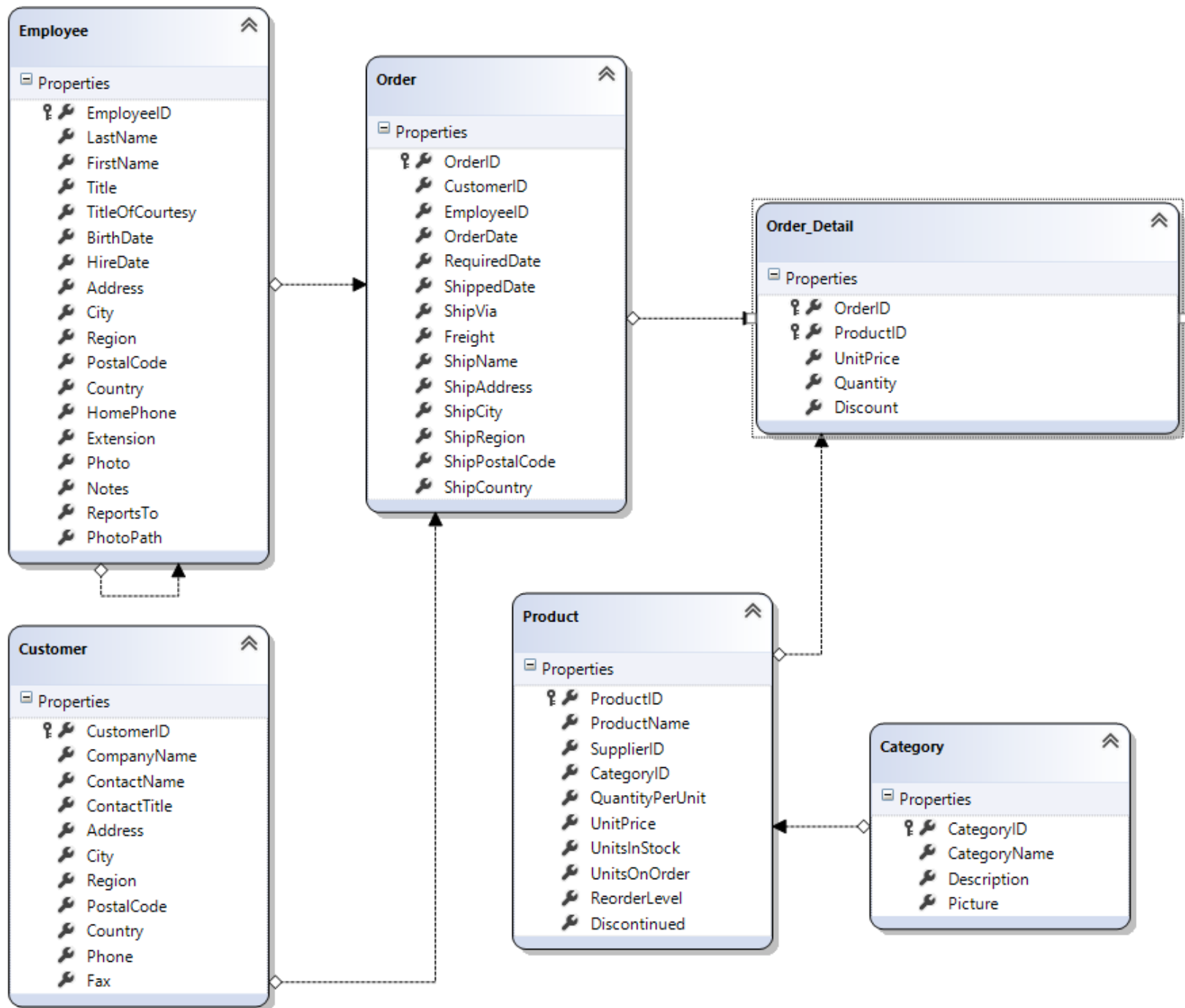
OK

Cancel



# LINQ to SQL

- Click phải ở tên Project trong Solution chọn Add => New Items => LINQ to SQL Classes; một file có đuôi .dbml sẽ được tạo ra.
- Chọn những bảng ở trong Server Explorer muốn ánh xạ vào ứng dụng và kéo thả vào giao diện của file .dbml vừa tạo.
- Như vậy các class có cùng tên với bảng trong CSDL đã được tạo ra.
- Một lớp kiểu **DataContext** sẽ được tạo ra cho phép truy vấn hay cập nhật các thay đổi trong class.



## Solution Explorer



Search Solution Explorer (Ctrl+;)



Solution 'LINQApp' (1 of 1 project)

▲ **LINQApp**

▷ Properties

▷ References

App.config

▲ DBNorthWind.dbml

DBNorthWind.dbml.layout

▲ DBNorthWind.designer.cs

▷ DBNorthWindDataContext

▷ Product

▷ Order

▷ Customer

▷ Employee

▷ Order\_Detail

▷ Category

▷ Program.cs

# LINQ to SQL

## Thực thi truy vấn:

- Sử dụng class DataContext để tạo nguồn dữ liệu.
- Truy vấn dữ liệu theo cú pháp truy vấn hoặc cú pháp phương thức.

```
DBNorthWindDataContext db = new DBNorthWindDataContext();  
//Select product name  
var product = from table in db.Products  
               select table.ProductName;  
  
foreach (var item in product)  
{  
    Console.WriteLine(item);  
}
```

# LINQ to SQL

- Lấy giá trị một số thuộc tính trong câu Select

```
var product = from p in db.Products
               select new
               {
                   IdProduct = p.ProductID,
                   NameProduct = p.ProductName
               };

foreach (var item in product)
{
    //Console.WriteLine(item);
    Console.WriteLine($"ID: {item.IdProduct}\t" +
                      $" Name: {item.NameProduct}");
}
```

# LINQ to SQL

- Thực hiện câu lệnh gom nhóm theo câu Select: Tính doanh thu của từng sản phẩm.

```
DBNorthWindDataContext db = new DBNorthWindDataContext();
//Tính doanh số từng khách hàng
var product = from pro in db.Products
               select new
               {
                   ID = pro.ProductID,
                   Name = pro.ProductName,
                   Revenue = pro.Order_Details.Sum(o=>o.UnitPrice * o.Quantity)
               };
foreach (var item in product)
{
    Console.WriteLine($"Product ID: {item.ID}\t " +
        $"Product Name: {item.Name}\t Revenue: {item.Revenue}");
}
```



# LINQ to SQL

- Lấy tên Product thuộc một loại Category dựa trên mối quan hệ một nhiều giữa Category và Product.

```
var product = from table in db.Products
               where table.Category.CategoryName == "Seafood"
               select table.ProductName;
foreach (var item in product)
{
    Console.WriteLine(item);
}
```

# LINQ to SQL

- Lấy những Product đã được bán trên 40 lần dựa trên mối quan hệ một nhiều giữa Product và OrderDetail.

```
var product = from pro in db.Products
               where pro.Order_Details.Count > 40
               select new
               {
                   Name = pro.ProductName,
                   Quantity = pro.Order_Details.Count
               };
foreach (var item in product)
{
    Console.WriteLine(item.Name + "\t" + item.Quantity);
}
```

# LINQ to SQL

 Sửa hoặc xóa dữ liệu trong CSDL với LINQ to SQL:

- Sau khi định nghĩa mô hình dữ liệu của CSDL với LINQ, các thuộc tính của mỗi lớp sẽ ánh xạ vào các cột tương ứng của bảng; mỗi đối tượng thuộc lớp sẽ biểu diễn một dòng trong bảng.
- LINQ to SQL tạo ra một lớp DataContext cung cấp các cách thức truy vấn và cập nhật lại dữ liệu.
- Để thực hiện việc cập nhật hay xóa dữ liệu, sẽ được thực hiện trên các đối tượng lấy ra từ LINQ to SQL.
- Sử dụng phương thức **SubmitChanges()** của lớp DataContext để hoàn tất việc thay đổi trên CSDL.

# LINQ to SQL

## Thêm một Product.

- `void InsertOnSubmit (TEntity entity)`: thêm một đối tượng.
- `void InsertAllOnSubmit<TSubEntity> (IEnumerable<TSubEntity> entities)` `where TSubEntity : TEntity`: thêm danh sách đối tượng.

```
DBNorthWindDataContext db = new DBNorthWindDataContext();  
Product pro = new Product();  
pro.ProductName = "Test";  
pro.UnitPrice = 300;  
pro.UnitsInStock = 1;  
pro.CategoryID = 1;  
pro.QuantityPerUnit = "Cai";  
  
db.Products.InsertOnSubmit(pro);  
db.SubmitChanges();
```

# LINQ to SQL

 Xóa một Product với ProductName = "Test"

- `void DeleteOnSubmit` (TEntity entity): Xóa từng đối tượng.
- `void DeleteAllOnSubmit`<TSubEntity> (IEnumerable<TSubEntity> entities) `where` TSubEntity : TEntity: Xóa danh sách đối tượng.

```
DBNorthWindDataContext db = new DBNorthWindDataContext();  
var delPro = from pro in db.Products  
              where pro.ProductName == "Test"  
              select pro;  
foreach (var item in delPro)  
{  
    db.Products.DeleteOnSubmit(item);  
}  
  
//db.Products.DeleteAllOnSubmit(delPro);  
db.SubmitChanges();
```

# LINQ to SQL

- 📖 Cập nhật UnitsInStock của Product thành 50 với các product có QuantityPerUnit là "Chai"

```
DBNorthWindDataContext db = new DBNorthWindDataContext();  
var product = db.Products.Where(pro => pro.QuantityPerUnit == "Chai");  
foreach (var pro in product)  
{  
    pro.UnitsInStock = 50;  
}  
db.SubmitChanges();
```

# LINQ to SQL

- 📖 Cập nhật ReorderLever của Product thành 0 với các product không có ai mua và UnitPrice >= 100

```
DBNorthWindDataContext db = new DBNorthWindDataContext();  
var product = from pro in db.Products  
               where pro.UnitPrice >= 100 && pro.Order_Details.Count == 0  
               select pro;  
foreach (var pro in product)  
{  
    pro.ReorderLevel = 0;  
}  
db.SubmitChanges();
```

# LINQ to SQL

- 📖 LINQ to SQL cho phép tận dụng các mối quan hệ của các bảng trong việc truy vấn và cập nhật dữ liệu.
- 📖 VD: thêm một Product mới và kết hợp nó với một category “Seafood”

```
DBNorthWindDataContext db = new DBNorthWindDataContext();  
Category cate = db.Categories.Single(c => c.CategoryName == "Seafood");
```

```
Product pro = new Product();  
pro.ProductName = "Shark";  
pro.UnitPrice = 300;  
pro.UnitsInStock = 1;
```

```
cate.Products.Add(pro);  
db.SubmitChanges();
```



# LINQ to SQL

Category

8	Seafood	Seaweed and fish
---	---------	------------------

Product

82	Shark	NULL	8
----	-------	------	---

# LINQ to SQL

VD: tạo mới một hóa đơn với 2 sản phẩm cho một khách hàng.

```
DBNorthWindDataContext db = new DBNorthWindDataContext();
//Lấy thông tin Product
Product pro1 = db.Products.Single(p => p.ProductName == "Bia");
Product pro2 = db.Products.Single(p => p.ProductName == "Nuoc ngot");
//Tạo hóa đơn
Order order = new Order{
    OrderDate = DateTime.Now,
    RequiredDate = DateTime.Now.AddDays(2),
    Freight = 26,
};
//Chi tiết
Order_Detail detail1 = new Order_Detail{
    Product = pro1,
    UnitPrice = 20,
    Quantity = 24
};
Order_Detail detail2 = new Order_Detail{
    Product = pro2,
    UnitPrice = 10,
    Quantity = 6
};
//Thêm vào hóa đơn
order.Order_Details.Add(detail1);
order.Order_Details.Add(detail2);
//Lấy thông tin Khách hàng
Customer cust = db.Customers.Single(c => c.CustomerID == "ALFKI");
//Thêm hóa đơn vào khách hàng
cust.Orders.Add(order);
db.SubmitChanges();
```

# LINQ to SQL

VD: tạo mới một hóa đơn với 2 sản phẩm cho một khách hàng.

## Order\_Details

OrderID	ProductID	UnitPrice	Quantity	Discount
11078	78	20.00	24	0
11078	80	10.00	6	0

## Orders

OrderID	CustomerID	OrderDate	RequiredDate	Freight
11078	ALFKI	2020-04-17 15:34:17.403	2020-04-19 15:34:17.403	26.00