



Cài đặt Sql Server Express và SSMS

CÀI ĐẶT SQL SERVER EXPRESS

CÀI ĐẶT SQL SERVER MANAGEMENT STUDIO

TẠO CƠ SỞ DỮ LIỆU MỚI TỪ SSMS

1. Kết nối SQL Server
2. Click phải vào nút Databases và chọn New Database...
3. Trong cửa sổ New Database, ô Database name nhập [Contacts](#). Các tham số khác để giá trị mặc định.
4. Sau khi hoàn thành chúng ta đã tạo được một cơ sở dữ liệu mới có tên Contacts trong nút Databases.

KẾT NỐI VISUAL STUDIO VỚI SQL SERVER DATABASE

1. Mở cửa sổ Server Explorer của Visual Studio (View => Server Explorer) và click phải vào Data Connections => chọn Add Connection.
2. Chọn Data Source là Microsoft SQL Server.
3. Nhập các thông tin như khi kết nối SSMS, bao gồm Server name và Authentication. Mỗi connection là một kết nối tới một cơ sở dữ liệu cụ thể.

CÀI ĐẶT LOCALDB

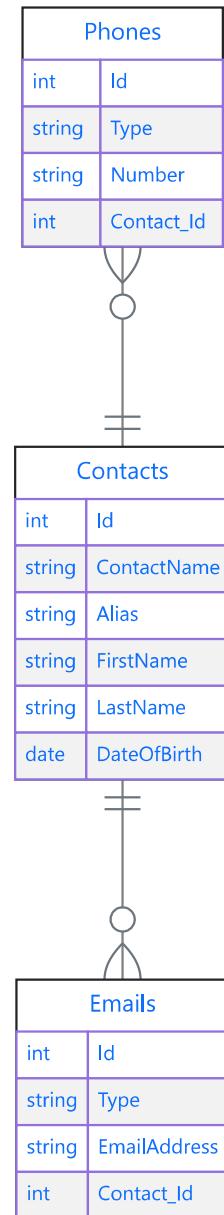
Thực hành: Tạo bảng, kết nối, đọc dữ liệu

Tạo bảng dữ liệu trong SSMS và SQL Server

Cấu trúc các bảng dữ liệu

Cơ sở dữ liệu Contacts đã tạo ở bài trước dùng để lưu trữ thông tin liên hệ của khách hàng. Mỗi bản ghi trong danh bạ chứa các thông tin:

- Tên liên hệ (contact name),
- Biệt danh (alias),
- Họ + đệm (first name),
- Tên (last name),
- Ngày sinh (birth day),
- Danh sách email, mỗi email trong danh sách bao gồm loại email (cá nhân, công việc, tổ chức) và địa chỉ email tương ứng.
- Danh sách số điện thoại, mỗi thông tin về điện thoại bao gồm loại (cá nhân, công việc), và số điện thoại tương ứng.



Tạo bảng dữ liệu từ SSMS

- Tạo các bảng

```
USE [Contacts]
GO
/***** Object: Table [dbo].[Contacts] *****/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[Contacts](
    [Id] [int] IDENTITY(1,1) NOT NULL,
    [ContactName] [nvarchar](max) NULL,
    [Alias] [nvarchar](max) NULL,
    [FirstName] [nvarchar](max) NULL,
    [LastName] [nvarchar](max) NULL,
    [DateOfBirth] [datetime] NOT NULL,
    CONSTRAINT [PK_dbo.Contacts] PRIMARY KEY CLUSTERED
(
    [Id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]
GO
/***** Object: Table [dbo].[Emails] *****/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[Emails](
    [Id] [int] IDENTITY(1,1) NOT NULL,
    [Type] [nvarchar](max) NULL,
    [EmailAddress] [nvarchar](max) NULL,
    [Contact_Id] [int] NULL,
    CONSTRAINT [PK_dbo.Emails] PRIMARY KEY CLUSTERED
(
    [Id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]
GO
/***** Object: Table [dbo].[Phones] *****/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[Phones](
    [Id] [int] IDENTITY(1,1) NOT NULL,
    [Type] [nvarchar](max) NULL,
    [Number] [nvarchar](max) NULL,
    [Contact_Id] [int] NULL,
    CONSTRAINT [PK_dbo.Phones] PRIMARY KEY CLUSTERED
(
    [Id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
```

```

) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]
GO
ALTER TABLE [dbo].[Emails] WITH CHECK ADD CONSTRAINT [FK_dbo.Emails_dbo.Contacts_Contact_Id] FOR
REFERENCES [dbo].[Contacts] ([Id])
GO
ALTER TABLE [dbo].[Emails] CHECK CONSTRAINT [FK_dbo.Emails_dbo.Contacts_Contact_Id]
GO
ALTER TABLE [dbo].[Phones] WITH CHECK ADD CONSTRAINT [FK_dbo.Phones_dbo.Contacts_Contact_Id] FOR
REFERENCES [dbo].[Contacts] ([Id])
GO
ALTER TABLE [dbo].[Phones] CHECK CONSTRAINT [FK_dbo.Phones_dbo.Contacts_Contact_Id]
GO

```

- Thêm dữ liệu

```

USE [Contacts]
GO
SET IDENTITY_INSERT [dbo].[Contacts] ON
INSERT INTO [dbo].[Contacts] ([Id], [ContactName], [Alias], [FirstName], [LastName], [DateOfBirth], [Address], [City], [State], [Country], [Phone], [Email], [Notes])
VALUES (1, N'John Doe', N'jdoe', N'John', N'Doe', '1980-01-01', N'123 Main St', N'Anytown', N'CA', N'USA', N'+1 555-1234', N'jdoe@example.com', N'Work')
INSERT INTO [dbo].[Contacts] ([Id], [ContactName], [Alias], [FirstName], [LastName], [DateOfBirth], [Address], [City], [State], [Country], [Phone], [Email], [Notes])
VALUES (2, N'Jane Smith', N'jsmith', N'Jane', N'Smith', '1985-05-20', N'456 Elm St', N'Anytown', N'CA', N'USA', N'+1 555-5678', N'jsmith@example.com', N'Home')
SET IDENTITY_INSERT [dbo].[Contacts] OFF
GO
SET IDENTITY_INSERT [dbo].[Emails] ON
INSERT INTO [dbo].[Emails] ([Id], [Type], [EmailAddress], [Contact_Id]) VALUES (1, N'Work', N'trungkhanh123@gmail.com', 1)
INSERT INTO [dbo].[Emails] ([Id], [Type], [EmailAddress], [Contact_Id]) VALUES (2, N'Home', N'donathan123@gmail.com', 2)
INSERT INTO [dbo].[Emails] ([Id], [Type], [EmailAddress], [Contact_Id]) VALUES (3, N'Work', N'obama@whitehouse.gov', 1)
INSERT INTO [dbo].[Emails] ([Id], [Type], [EmailAddress], [Contact_Id]) VALUES (4, N'Home', N'barack@whitehouse.gov', 1)
SET IDENTITY_INSERT [dbo].[Emails] OFF
SET IDENTITY_INSERT [dbo].[Phones] ON
INSERT INTO [dbo].[Phones] ([Id], [Type], [Number], [Contact_Id]) VALUES (1, N'Work', N'+1 555-1234', 1)
INSERT INTO [dbo].[Phones] ([Id], [Type], [Number], [Contact_Id]) VALUES (2, N'Home', N'+1 555-5678', 2)
SET IDENTITY_INSERT [dbo].[Phones] OFF
GO

```

Thực hành 1

```

using System;
using System.Data.SqlClient;
namespace Example01
{
    internal class Program
    {
        private static void Main(string[] args)
        {
            // object đảm nhiệm việc kết nối
            var connection = new SqlConnection

```

```

    {
        // chuỗi ký tự chứa tham số phục vụ kết nối
        ConnectionString = @"""Data Source=xxx;Initial Catalog=Contacts;Integrated Security=True""";
    };
    // object đảm nhiệm việc thực thi truy vấn
    var command = new SqlCommand()
    {
        Connection = connection,
        CommandText = "SELECT COUNT(*) FROM CONTACTS"
    };

    // thử mở kết nối
    connection.Open();
    // thực thi truy vấn và lấy kết quả
    var res = (int)command.ExecuteScalar();
    // đóng kết nối
    connection.Close();
    Console.WriteLine($"{res} contacts found in the database");
    Console.ReadLine();
}
}
}

```

Thực hành 2

```

using System;
using System.Data;
using System.Data.SqlClient;
namespace Example02
{
    internal class Program
    {
        private static void Main(string[] args)
        {
            var connectionString = @"""Data Source=xxx;Initial Catalog=Contacts;Integrated Security=True""";
            var connection = new SqlConnection(connectionString);
            var command = new SqlCommand("SELECT * FROM CONTACTS", connection);
            connection.Open();
            var dataReader = command.ExecuteReader(CommandBehavior.CloseConnection);
            if (dataReader.HasRows)
            {
                while (dataReader.Read())
                {
                    var id = dataReader.GetInt32(0);
                    var contactName = dataReader.GetString(1);
                    var alias = dataReader.GetString(2);
                    Console.WriteLine($"[{id}] {contactName} ({alias})");
                }
            }
        }
    }
}

```

```
        }
    }
    Console.ReadLine();
}
}
```

Thực hành 3

```
using System;
using System.Data;
using System.Data.SqlClient;
namespace Example03
{
    internal class Program
    {
        private static void Main(string[] args)
        {
            var connectionString = @"Data Source=xxx;Initial Catalog=Contacts;Integrated Security=True";
            var dataTable = new DataTable();
            using (var connection = new SqlConnection(connectionString))
            {
                var command = connection.CreateCommand();
                command.CommandText = "SELECT * FROM CONTACTS";
                var dataAdapter = new SqlDataAdapter(command);
                dataAdapter.Fill(dataTable);
            }
            foreach (DataRow r in dataTable.Rows)
            {
                var id = (int)r["Id"];
                var contactName = (string)r["ContactName"];
                var alias = (string)r["Alias"];
                Console.WriteLine($"[{id}] {contactName} ({alias})");
            }
            Console.ReadLine();
        }
    }
}
```

Giới thiệu kiến trúc ADO.NET, data source, data provider

Vai trò và vị trí của ADO.NET

Dữ liệu của chương trình là thành phần quan trọng hàng đầu và có thể được lưu trữ (và truy xuất) theo nhiều cách khác nhau. Có hai hướng đi chính:

1. tự mình quản lý dữ liệu.
2. sử dụng một chương trình hoặc kiến trúc chuyên dụng để quản lý dữ liệu, thường gọi chung là database.

Đối với hướng sử dụng database cũng có hai lựa chọn:

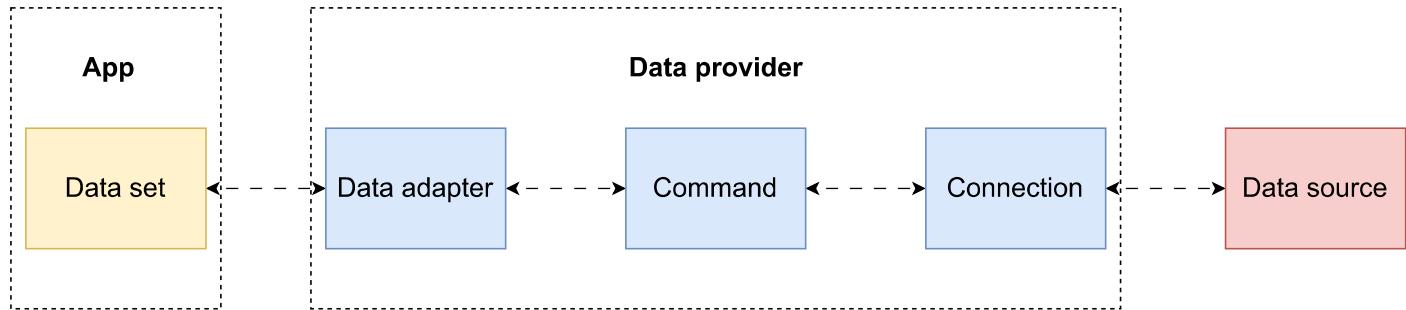
1. sử dụng database theo kiến trúc client-server.
2. sử dụng database dựa trên file.

Các database lưu trữ thông tin theo dạng của riêng mình. Thường gặp nhất là dạng bảng, nhưng cũng có thể là dạng xml hoặc json. Trong khi đó, chương trình lưu trữ thông tin dưới dạng chuỗi phân cấp của các đối tượng.

ADO.NET là một bộ phận của .NET framework. Nó đóng vai trò là một bộ công cụ và một lớp trung gian giúp chương trình tương tác với nguồn dữ liệu.

Kiến trúc ADO.NET

ADO.NET chia làm hai nhóm: **Disconnected** và **Connected**, tương tác với nhau thông qua Data Adapter



Thành phần Connected

Thành phần Connected bao gồm các object chịu trách nhiệm tương tác trực tiếp với nguồn dữ liệu: connection, command, parameter.

Để ý rằng, ở đây chúng ta chỉ nói về object mà không nói tới class cụ thể nào do ADO.NET có thể làm việc với nhiều loại nguồn dữ liệu. Ứng với mỗi loại nguồn dữ liệu sẽ phải sử dụng nhóm class riêng biệt. Tuy nhiên các class này đều phải thực thi một interface chung.

Connection chịu trách nhiệm cho việc kết nối tới nguồn dữ liệu. Các công việc chính connection đảm nhiệm là mở/đóng kết nối, kiểm tra tình trạng kết nối. Tất cả các thành phần còn lại của nhóm Connected đều hoạt động trên connection.

Command là thành phần chịu trách nhiệm thực thi các truy vấn như đọc, ghi, cập nhật, xóa dữ liệu (gọi chung là nhóm lệnh CRUD – Create – Retrieve – Update – Delete). Command cũng thể chịu trách nhiệm làm

việc với cấu trúc của nguồn dữ liệu, ví dụ thay đổi cấu trúc bảng. Command hoạt động trên một connection cụ thể.

Parameter chịu trách nhiệm truyền tham số một cách linh hoạt và an toàn cho các truy vấn. Parameter hoạt động trên các Command và giúp đưa tham số cho các Command.

Thành phần này phụ thuộc vào loại nguồn dữ liệu. Ví dụ, để làm việc với SQL server phải tạo ra object chuyên dụng cho loại nguồn dữ liệu này (từ class SqlConnection); đối với Oracle, object sẽ được tạo ra từ class OracleConnection.

Thành phần Disconnected

Thành phần này chịu trách nhiệm tạo ra một bản sao của (một phần) cơ sở dữ liệu trong bộ nhớ chương trình. Bản sao này cho phép chương trình có thể làm việc với dữ liệu mà không cần duy trì kết nối liên tục tới nguồn dữ liệu.

Để tạo ra "bản sao" của cơ sở dữ liệu, thành phần Disconnected chứa các lớp mô phỏng cấu trúc của cơ sở dữ liệu, bao gồm:

- `DataSet` (mô phỏng cả cơ sở dữ liệu)
- `DataTable` (mô phỏng bảng dữ liệu)
- `DataRow` (dòng dữ liệu)
- `DataColumn` (cột dữ liệu)
- `DataView` (tương tự như view của Sql Server)
- `DataRelation` (quan hệ giữa các bảng)

Thành phần Disconnected không phụ thuộc vào nguồn dữ liệu cụ thể nào. Dữ liệu từ bất kỳ nguồn dữ liệu nào đều có thể đổ vào `DataSet`, `DataTable`.

Thành phần Disconnected không thể tự mình làm việc trực tiếp với nguồn dữ liệu. Thay vào đó, nó phải tương tác với thành phần Connected thông qua Data Adapter để thực hiện các thao tác CRUD.

Data Adapter

Data Adapter là thành phần trung gian đặc biệt giúp thành phần Disconnected làm việc với thành phần Connected. Data Adapter hỗ trợ tự động thực hiện các thao tác CRUD với nguồn dữ liệu khi cần thiết.

Data Provider

ADO.NET chia thành phần Connected theo từng loại nguồn dữ liệu. Ví dụ,

- để làm việc với SQL Server sẽ cần một nhóm class riêng (nằm trong không gian tên System.Data.SqlClient).
- để làm việc Oracle cũng cần một nhóm class như vậy (trong không gian tên System.Data.OracleClient).

Một nhóm class dành cho thành phần Connected để làm việc với một loại nguồn dữ liệu cụ thể như vậy được gọi là Data Provider.

Kết nối tới SQL Server

Thực hành: kết nối đến cơ sở dữ liệu Sql Server

```
using System;
using System.Data.SqlClient;
using System.Data;
namespace ConnectToSql
{
    class Program
    {
        static void Main(string[] args)
        {
            var connectionString = @"Data Source=xxx;Initial Catalog=Contacts;Integrated Security=True";
            var connection = new SqlConnection
            {
                ConnectionString = connectionString
            };
            try
            {
                connection.Open();
                if(connection.State == ConnectionState.Open)
                {
                    Console.WriteLine("Connection opened successfully!");
                }
            }
            catch (Exception)
            {
                if(connection.State != ConnectionState.Open)
                {
                    Console.WriteLine("Failed to open the connection");
                }
            }
            finally
            {
                if(connection.State == ConnectionState.Open)
                {
                    connection.Close();
                }
                connection.Dispose();
            }
            Console.ReadLine();
        }
    }
}
```

```
    }  
}
```

Phương thức `Open` của lớp `SqlConnection` thực hiện việc mở kết nối. Khi kết thúc phiên làm việc với cơ sở dữ liệu cần gọi phương thức `Close` để đóng kết nối.

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

Thuộc tính `State` của `SqlConnection` trả về thông tin về trạng thái của kết nối ở thời điểm hiện tại. Thuộc tính này có giá trị thuộc kiểu liệt kê `ConnectionState` (nằm trong không gian tên `System.Data`).

```
if(connection.State == ConnectionState.Open) {...}
```

ConnectionString – chuỗi tham số hỗ trợ kết nối

Connection string là một chuỗi ký tự chứa các cặp khóa – giá trị. Mỗi cặp khóa – giá trị là một tham số phục vụ kết nối. Nói chung, chuỗi kết nối có dạng như sau:

```
"parametername1=parametervalue1;parametername2=parametervalue2;..."
```

Phương pháp xác thực

```
// chuỗi connectionstring chứa thông tin về cách xác thực sử dụng windows authentication  
var connectionString1 = "Integrated Security=true";  
var connectionString2 = "Integrated Security=SSPI";  
// chuỗi connectionstring chứa user id và mật khẩu (phương pháp xác thực Sql Server)  
// sa là tên tài khoản super admin, nhớ thay bằng mật khẩu của mình  
var connectionString3 = "User ID=sa; Password=1234567890";
```

Chỉ định host và instance của Sql Server

```
// kết nối tới instance mặc định trên máy cục bộ  
var connectionString4 = @"Server=.; Integrated Security=SSPI";  
var connectionString5 = @"Server=(local); Integrated Security=SSPI";  
var connectionString6 = @"Data Source=localhost; Integrated Security=SSPI";  
var connectionString7 = @"Data Source=local-pc; Integrated Security=SSPI";//nhớ thay local-pc bằng  
// kết nối tới instance xxx trên máy cục bộ  
var connectionString8 = @"Server=.\xxx; Integrated Security=SSPI";  
var connectionString9 = @"Server=(local)\xxx; Integrated Security=SSPI";  
var connectionString10 = @"Data Source=localhost\xxx; Integrated Security=SSPI";  
var connectionString11 = @"Data Source=local-pc\xxx; Integrated Security=SSPI";//nhớ thay local-pc
```

Chỉ định cơ sở dữ liệu

```
var connectionString12 = @"Data Source=.\xxx; Integrated Security=SSPI; Initial Catalog=Contacts";
```

Lưu ý:

1. thứ tự của các cặp tham số trong connectionstring không quan trọng
2. không phân biệt chữ hoa thường trong từ khóa và giá trị

Connection string để kết nối tới file cơ sở dữ liệu sử dụng LocalDb

```
Data Source=(localdb)\MSSQLLocalDB; AttachDbFilename=C:\Users\ Laptop\ Library.mdf; Integrated Security=True;
```

Tạo – lưu trữ – truy xuất ConnectionString cho chương trình

Lớp ConnectionStringBuilder – tạo connection string

```
SqlConnectionStringBuilder connectionStringBuilder = new SqlConnectionStringBuilder();  
connectionStringBuilder.DataSource = "(local)";  
connectionStringBuilder.InitialCatalog = "Contacts";  
connectionStringBuilder.IntegratedSecurity = true;  
var connectionString = connectionStringBuilder.ToString();
```

Tạo connection string bằng file udl

Lưu và truy xuất connection string từ file cấu hình

Để sử dụng App.config với connection string chúng ta thực hiện các bước sau:

Bước 1. Thay đổi file [App.config](#) bằng cách thêm node , trong đó thêm node <add ... /> như dưới đây

```
<?xml version="1.0" encoding="utf-8" ?>  
<configuration>  
    <startup>  
        <supportedRuntime version="v4.0" sku=".NETFramework, Version=v4.6.1" />  
    </startup>  
    <connectionStrings>  
        <add name="my connection string" connectionString="Data Source=.\xxx;Initial Catalog=Contacts" />  
    </connectionStrings>  
</configuration>
```

Bước 2. Tham chiếu project tới thư viện System.Configuration (click phải node References => Add Reference => tìm System.Configuration trong nhóm Assemblies => OK).

Bước 3. Truy xuất chuỗi liên kết từ code C# như sau:

```
var connectionString = System.Configuration.ConfigurationManager.ConnectionStrings["my connection
```

Sử dụng SqlConnection để xử lý kết nối tới Sql Server

Lớp SqlConnection

ADO.NET hỗ trợ kết nối tới nguồn dữ liệu bằng cách sử dụng một object của class được thiết kế riêng để làm việc với cơ sở dữ liệu tương ứng. Class này là một phần của *data provider* được tạo riêng dành cho nguồn dữ liệu đó.

Để kết nối tới cơ sở dữ liệu SQL Server cần sử dụng một object từ lớp [SqlConnection](#). Class này là một phần của data provider dành cho SQL Server. Lớp SqlConnection nằm trong namespace [System.Data.SqlClient](#).

Có một số cách để khởi tạo object của class này:

```
var connection1 = new SqlConnection();
var connection2 = new SqlConnection(@"Data Source=.\xxx;Initial Catalog=Contacts;Integrated Security=True");
var connection3 = new SqlConnection { ConnectionString = @"Data Source=.\xxx;Initial Catalog=Contac
```

Các cách thức khởi tạo này đem lại cùng một kết quả. Ở cách khởi tạo thứ hai và thứ ba chúng ta cung cấp một chuỗi ký tự chứa tham số phục vụ kết nối. Nên lưu ý rằng, trước khi mở liên kết bắt buộc phải cung cấp thông tin về chuỗi kết nối này cho object của SqlConnection.

Trong phần tiếp theo chúng ta sẽ xem xét các phương thức và thuộc tính quan trọng của lớp SqlConnection.

Phương thức của lớp SqlConnection

[Open](#): đây là phương thức quan trọng hàng đầu của lớp SqlConnection. Phương thức này mở kết nối vật lý tới cơ sở dữ liệu. Object chỉ sẵn sàng để sử dụng với cơ sở dữ liệu sau khi gọi phương thức Open. Trước khi gọi phương thức này bắt buộc phải cung cấp một connectionstring phù hợp.

[Close](#): phương thức này thực hiện công việc ngược lại so với Open – đóng kết nối tới cơ sở dữ liệu nếu kết nối đang mở. Một khi gọi phương thức Close, kết nối vật lý sẽ không bị xóa bỏ thực sự mà đưa vào một "kho lưu tạm" gọi là *Connection Pool* để có thể tái sử dụng về sau. Một connection sau khi gọi Close sẽ có thể được sử dụng lại.

Lưu ý, một connection nên mở càng muộn càng tốt, và nên đóng càng sớm càng tốt.

Dispose: tương tự như Close, Dispose sẽ đóng kết nối và đưa nó vào pool. Dispose tự nó sẽ gọi tới Close. Sự khác biệt là Dispose đồng thời xóa bỏ tất cả các trạng thái và thông tin của connection (như ConnectionString). Một connection khi bị Dispose thì chương trình không thể tái sử dụng.

ChangeDatabase: cho phép chuyển đổi cơ sở dữ liệu sẽ được sử dụng. Như ở trên chúng ta đã biết, khi tạo connectionstring, chúng ta có thể kết nối tới bất kỳ cơ sở dữ liệu nào ngay từ đầu hoặc chỉ kết nối tới server. Để thực hiện các thao tác với dữ liệu, ta phải chọn cơ sở dữ liệu cụ thể bằng phương thức ChangeDatabase. Dĩ nhiên, chúng ta có thể chuyển đổi cơ sở dữ liệu bằng cách tạo liên kết mới. Tuy nhiên, ChangeDatabase hiệu quả hơn nhiều.

CreateCommand: giúp tạo ra một object của lớp SqlCommand – class giúp thực hiện các truy vấn trên cơ sở dữ liệu.

Thuộc tính của lớp SqlConnection

ConnectionString: đây là thuộc tính quan trọng hàng đầu của SqlConnection. Thuộc tính này chứa chuỗi liên kết mà chúng ta đã xem xét rất kỹ ở các phần trên. Thuộc tính này bắt buộc phải có giá trị phù hợp trước khi gọi phương thức Open.

ConnectionTimeout: chứa giá trị thời gian (tính bằng giây) tối đa để thử kết nối tới server. Quá thời gian này mà chưa kết nối thành công, việc kết nối sẽ bị hủy bỏ và phát ra **exception**.

Database: chứa thông tin (chỉ đọc) về cơ sở dữ liệu hiện tại đang được sử dụng (nếu liên kết tới cơ sở dữ liệu đang mở).

State: chứa thông tin về trạng thái của hiện tại của liên kết. Thuộc tính này có giá trị thuộc kiểu liệt kê **ConnectionState** (nằm trong không gian tên **System.Data**). Hai giá trị quan trọng mà chúng ta đã biết là ConnectionState.Open và ConnectionState.Closed.

Connection Pooling

Connection pooling là một loại kỹ thuật cho phép tạo ra và duy trì một số các kết nối sử dụng chung để tăng hiệu suất cho các ứng dụng. Việc tăng hiệu suất này được thực hiện thông qua tái sử dụng các kết nối khi có yêu cầu thay vì việc tạo kết nối mới.

Tại sao lại cần Connection Pooling? Lý do là vì việc tạo và hủy các kết nối tới cơ sở dữ liệu mất rất nhiều thời gian. Thông thường phải mất từ 1 đến 3s để thiết lập kết nối tới cơ sở dữ liệu vì trong đó phải thực hiện các bước như giao tiếp với server, chứng thực, và nhiều tác vụ khác. Do đó, việc đóng mở kết nối liên tục sẽ ảnh hưởng rất lớn tới hiệu năng của ứng dụng và tải của server.

Trong ADO.NET, Connection Pooling *được sử dụng mặc định*. Đối với SqlConnection, khi gọi một trong hai phương thức Close và Dispose (đã trình bày ở trên) sẽ tự động đưa kết nối đó vào pool (tạm dịch là *vùng lưu trữ liên kết tạm thời*) mà không hủy bỏ hoàn toàn liên kết. Khi gấp lệnh Open tiếp theo, một kết nối trong pool sẽ được sử dụng (nếu có) thay vì tạo ra liên kết mới.

Thực thi truy vấn SQL, SqlCommand, SqlParameter

Truy vấn SQL

Lớp SqlCommand

SqlCommand (tên đầy đủ là `System.Data.SqlClient.SqlCommand`) là class chịu trách nhiệm thực thi truy vấn trên một kết nối tới cơ sở dữ liệu Sql Server.

Khởi tạo object

Có một số cách khởi tạo object của lớp SqlCommand:

```
var command1 = new SqlCommand();
// cung cấp chuỗi truy vấn khi khởi tạo
var commandText = "Select * from Contacts";
var command2 = new SqlCommand(commandText);
// cung cấp chuỗi truy vấn + object connection khi khởi tạo
var connection = new SqlConnection();
var command3 = new SqlCommand(commandText, connection);
// cũng có thể tạo command trực tiếp từ connection
var connection = new SqlConnection();
var command = connection.CreateCommand();
```

Một số thuộc tính của SqlCommand

`CommandText`: chứa chuỗi truy vấn SQL cần thực thi

`Connection`: chứa object của connection được sử dụng để kết nối tới cơ sở dữ liệu

`CommandType`: xác định xem mình cần thực hiện truy vấn SQL (giá trị `CommandType.Text`) hay cần gọi hàm stored procedure (giá trị `CommandType.StoredProcedure`) qua object của `SqlCommand`. Giá trị mặc định là `Text`.

`Parameters`: danh sách tham số được sử dụng trong truy vấn. Nội dung này sẽ trình bày ở một phần riêng cuối bài.

Hai thông tin quan trọng nhất mà object `SqlCommand` cần biết là object của class `SqlConnection` và chuỗi truy vấn Sql. Hai thông tin này phải được cung cấp cho object `SqlCommand` trước khi gọi lệnh thực thi bất kỳ truy vấn nào.

Một số phương thức quan trọng của SqlCommand

`ExecuteNonQuery()`: chuyên để thực thi các truy vấn không trả về dữ liệu (INSERT, UPDATE, DELETE).

`ExecuteScalar()`: thực thi các truy vấn trả lại MỘT giá trị duy nhất, thường là kết quả của truy vấn Aggregate (SELECT COUNT|MIN|MAX|AVG).

`ExecuteReader()`: thực thi các truy vấn trả lại TẬP HỢP giá trị, như các truy vấn SELECT thông thường.

Xử lý kết quả truy vấn của SqlCommand

Đối với việc nhận kết quả, tùy thuộc vào từng loại truy vấn sẽ những kỹ thuật khác nhau.

Như bạn đã biết, ngôn ngữ SQL có 4 loại truy vấn dữ liệu chính: INSERT, UPDATE, DELETE, và SELECT. Trong đó, INSERT, UPDATE, DELETE không trả lại kết quả. Phương thức ExecuteNonQuery() dùng để thực thi các truy vấn này. Kết quả thực hiện của phương thức là số bản ghi chịu ảnh hưởng của truy vấn.

Riêng đối với SELECT, tùy từng truy vấn phải sử dụng cách lưu trữ dữ liệu phù hợp.

Đối với các truy vấn kiểu SELECT COUNT|MIN|MAX|AVG, kết quả trả về chỉ là một giá trị đơn. Phương thức ExecuteScalar sẽ lưu kết quả vào một biến kiểu Object. Bạn có thể cast nó sang kiểu phù hợp để sử dụng tiếp.

Đối với các truy vấn SELECT khác, kết quả trả về là một tập hợp dữ liệu. Một phần kết quả của các truy vấn này được lưu (tạm) trong một object [SqlDataReader](#).

SqlDataReader hoạt động theo kiểu forward-only và read-only. SqlDataReader cho phép đọc qua từng hàng trong tập hợp dữ liệu theo một chiều từ đầu đến cuối (forward-only). Không thể đọc theo chiều ngược lại. SqlDataReader chỉ cho phép đọc dữ liệu ra mà không cho phép sửa dữ liệu (read-only). Do đó, nếu muốn lưu trữ kết quả để sử dụng lâu dài, bạn phải dùng một cơ chế lưu trữ khác.

Cách thức làm việc cụ thể của SqlDataReader sẽ xem xét ở phần ví dụ dưới đây.

Thực thi truy vấn Sql với SqlCommand

Thực thi truy vấn Aggregate

```
using System;
using System.Data.SqlClient;
namespace P01_Scalar
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.Title = "Retrieve scalar value";
            // lưu ý thay connectionString của mình
            var connectionString = @"Data Source =.\xxx; Initial Catalog = Contacts; Integrated Security = True";
            // sử dụng cấu trúc using để connection tự động close sau khi kết thúc khối code
            using (var connection = new SqlConnection(connectionString))
            {
                // truy vấn SQL
                var commandText = "SELECT COUNT(*) FROM CONTACTS";
                // khởi tạo object của lớp SqlCommand
                var command = new SqlCommand(commandText, connection);
                // lưu ý phải mở kết nối trước khi thực thi truy vấn
                connection.Open();
                // thực thi truy vấn để lấy kết quả scalar
                var count = (int)command.ExecuteScalar();
            }
        }
    }
}
```

```
        Console.WriteLine($"{count} contacts found in the database");
    }
    Console.ReadKey();
}
}
```

Thực thi truy vấn Select

```
using System;
using System.Data;
using System.Data.SqlClient;
namespace P02_DataReader
{
    class Program
    {
        private static void Main(string[] args)
        {
            Console.Title = "Retrieve data set";
            var connectionString = @"Data Source=.\xxx;Initial Catalog=Contacts;Integrated Security=True";
            using (var connection = new SqlConnection(connectionString))
            using (var command = new SqlCommand("SELECT * FROM CONTACTS", connection))
            {
                connection.Open();
                var sqlDataReader = command.ExecuteReader(CommandBehavior.CloseConnection);
                if (sqlDataReader.HasRows)
                {
                    while (sqlDataReader.Read())
                    {
                        var id = sqlDataReader.GetInt32(0);
                        var contactName = sqlDataReader.GetString(1);
                        var alias = sqlDataReader.GetString(2);
                        Console.WriteLine($"[{id}] {contactName} ({alias})");
                    }
                }
            }
            Console.ReadLine();
        }
    }
}
```

Sử dụng phương thức ExecuteReader

Sử dụng SqlDataReader

Sử dụng phép toán indexer và số thứ tự của ô trên object của SqlDataReader:

```
var id = (int) sqlDataReader[0];
var contactName = sqlDataReader[1] as string;
var alias = sqlDataReader[2] as string;
```

Sử dụng tên trường trong phép toán indexer:

```
var id = (int) sqlDataReader["Id"];
var contactName = sqlDataReader["ContactName"] as string;
var alias = sqlDataReader["Alias"] as string;
```

Thực thi truy vấn INSERT – UPDATE – DELETE

```
using System;
using System.Data.SqlClient;
namespace P03_InsertUpdateDelete
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.Title = "Insert - Update - Delete";
            var connectionString = @"Data Source=.;Initial Catalog=Contacts;Integrated Security=True";
            using (var connection = new SqlConnection(connectionString))
            using (var command = new SqlCommand { Connection = connection })
            {
                connection.Open();
                Console.WriteLine("Before inserting:");
                Retrieve(command);
                var insertQuery = "INSERT INTO [dbo].[Contacts] ([ContactName], [Alias], [FirstName], [LastName]) VALUES ('George Jr. Bush', 'GJB', 'George', 'Bush')";
                command.CommandText = insertQuery;
                var count = command.ExecuteNonQuery();
                Console.WriteLine($"\\r\\n{count} record inserted!");
                Console.WriteLine("\\r\\nAfter inserting:");
                Retrieve(command);
                var updateQuery = "UPDATE [dbo].[Contacts] SET [ContactName] = 'George Jr. Bush' WHERE [ContactName] like '%Bush%'";
                command.CommandText = updateQuery;
                count = command.ExecuteNonQuery();
                Console.WriteLine($"\\r\\n{count} record updated!");
                Console.WriteLine("\\r\\nAfter updating:");
                Retrieve(command);
                var deleteQuery = "DELETE FROM [dbo].[Contacts] WHERE [ContactName] like '%Bush%'";
                command.CommandText = deleteQuery;
                count = command.ExecuteNonQuery();
                Console.WriteLine($"\\r\\n{count} record deleted!");
                Console.WriteLine("\\r\\nAfter deleting:");
                Retrieve(command);
            }
            Console.ReadLine();
        }
    }
}
```

```

static void Retrieve(SqlCommand command)
{
    command.CommandText = "SELECT * FROM CONTACTS";
    var sqlDataReader = command.ExecuteReader();
    if (sqlDataReader.HasRows)
    {
        while (sqlDataReader.Read())
        {
            var id = (int)sqlDataReader["Id"];
            var contactName = sqlDataReader["ContactName"] as string;
            var alias = sqlDataReader["Alias"] as string;
            Console.WriteLine($"[{id}] {contactName} ({alias})");
        }
    }
    sqlDataReader.Close();
}
}

```

Tham số trong truy vấn SQL, SqlParameter

Vấn đề tạo truy vấn từ dữ liệu người dùng

```

var contactName = Console.ReadLine();
var fistName = Console.ReadLine();
var lastName = Console.ReadLine();
var commandText = $"INSERT INTO [dbo].[Contacts] ([ContactName], [FirstName], [LastName]) VALUES ({contactName}, {firstName}, {lastName})";
// các thao tác còn lại bỏ qua

```

Lớp SqlParameter – ví dụ minh họa

```

using System;
using System.Data.SqlClient;
using System.Data;
namespace P04_Parameter
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.Title = "Parameters";
            var connectionString = @"Data Source=.;Initial Catalog=Contacts;Integrated Security=True";
            while (true)
            {
                Console.WriteLine("Create new contact:");

```

```
Console.WriteLine("Contact name: ");
var contactName = Console.ReadLine();
Console.WriteLine("First name: ");
var firstName = Console.ReadLine();
Console.WriteLine("Last name: ");
var lastName = Console.ReadLine();
// Viết chuỗi truy vấn bình thường, tuy nhiên, ở vị trí nào cần tham số thì viết @
var query = "INSERT INTO [dbo].[Contacts] ([ContactName], [FirstName], [LastName], [BirthDay]) VALUES (@ContactName, @FirstName, @LastName, @BirthDay)";
// Tạo các object SqlParameter để chứa tham số. Có nhiều overload khác nhau của hàm
// Lưu ý rằng, phần tên tham số của object phải trùng khớp với tên đã sử dụng trong câu lệnh
var contactNameParam = new SqlParameter("ContactName", contactName);
var firstNameParam = new SqlParameter("FirstName", firstName);
// hoặc sử dụng cách khởi tạo dùng property
var birthDayParam = new SqlParameter("BirthDay", birthDay);
{
    DbType = DbType.Date,
    ParameterName = "DoB",
    Value = DateTime.Now
};
using (var connection = new SqlConnection(connectionString))
using (var command = new SqlCommand { Connection = connection })
{
    command.CommandText = query;
    // lần lượt thêm từng object SqlParameter vào danh sách Parameters của command
    command.Parameters.Add(contactNameParam);
    command.Parameters.Add(firstNameParam);
    command.Parameters.Add(birthDayParam);
    // thậm chí có thể trực tiếp thêm cặp tham số - giá trị theo cách này mà không cần
    // command.AddWithValue("LastName", lastName);
    connection.Open();
    var count = command.ExecuteNonQuery();
    Console.WriteLine($"{count} contact inserted!");
    Retrieve(command);
}
Console.ReadKey();
}
}
static void Retrieve(SqlCommand command)
{
    command.CommandText = "SELECT * FROM CONTACTS";
    var sqlDataReader = command.ExecuteReader();
    if (sqlDataReader.HasRows)
    {
        while (sqlDataReader.Read())
        {
            var id = (int)sqlDataReader["Id"];
            var contactName = sqlDataReader["ContactName"] as string;
            var alias = sqlDataReader["Alias"] as string;
            Console.WriteLine($"[{id}] {contactName} ({alias})");
        }
    }
}
```

```
    sqlDataReader.Close();
}
}
}
```