Contents

[**[ADO.NET Tutorial] Lesson 01: Giới thiệu về ADO.NET** 2](#_Toc72134476)

[**Giới thiệu** 2](#_Toc72134477)

[**Data Provider** 2](#_Toc72134478)

[**Các đối tượng của ADO.NET** 3](#_Toc72134479)

[**Tổng kết** 4](#_Toc72134480)

[[ADO.NET Tutorial] Lesson 02: Đối tượng SqlConnection 6](#_Toc72134481)

[Giới thiệu 6](#_Toc72134482)

[Tạo một đối tượng SqlConnection 7](#_Toc72134483)

[Sử dụng SqlConnection 8](#_Toc72134484)

[Tổng kết 11](#_Toc72134485)

[[ADO.NET Tutorial] Lesson 03: Đối tượng SqlCommand 12](#_Toc72134486)

[Giới thiệu 12](#_Toc72134487)

[Tạo một đối tượng SqlCommand 12](#_Toc72134488)

[Querying Data – Truy vấn dữ liệu 12](#_Toc72134489)

[Inserting Data – Chèn dữ liệu 13](#_Toc72134490)

[Updating Data – Cập nhật dữ liệu 14](#_Toc72134491)

[Deleting Data – Xóa dữ liệu 14](#_Toc72134492)

[Lấy một giá trị đơn 15](#_Toc72134493)

[Kết hợp tất cả với nhau 16](#_Toc72134494)

[Tổng kết 24](#_Toc72134495)

[[ADO.NET Tutorial] Lesson 04: Đọc dữ liệu với SqlDataReader 25](#_Toc72134496)

[Giới thiệu 25](#_Toc72134497)

[Tạo một đối tượng SqlDataReader 26](#_Toc72134498)

[Đọc dữ liệu 26](#_Toc72134499)

[Hoàn tất 27](#_Toc72134500)

[Tổng kết 30](#_Toc72134501)

[[ADO.NET Tutorial] Lesson 05: Làm việc với Disconnected Data – DataSet và SqlDataAdapter 31](#_Toc72134502)

[Giới thiệu 31](#_Toc72134503)

[Tạo đối tượng DataSet 32](#_Toc72134504)

[Tạo một SqlDataAdapter 32](#_Toc72134505)

[Đổ dữ liệu vào DataSet 33](#_Toc72134506)

[Sử dụng DataSet 33](#_Toc72134507)

[Cập nhật thay đổi 34](#_Toc72134508)

[Kết hợp tất cả lại 34](#_Toc72134509)

[Tổng kết 37](#_Toc72134510)

[**[ADO.NET Tutorial] Lesson 01: Giới thiệu về ADO.NET**](https://yinyangit.wordpress.com/2011/07/31/ado-net-tutorial-lesson-01-introduction-to-ado-net/)

[](https://yinyangit.wordpress.com/2011/07/31/ado-net-tutorial-lesson-01-introduction-to-ado-net/ado-net-data-access-architecture/)**(The C# Station ADO.NET Tutorial)**  
Lesson này sẽ giới thiệu các khái niệm chính của ADO.NET và các đối tượng mà bạn sẽ học trong các bài tiếp theo. Ngoài việc hiểu rõ hơn về khái niệm ADO.NET, bạn sẽ được biết thêm về các khái niệm cơ bản khác như data provider và các đối tượng connection, command, DataReader, DataSet, DataAdapter.

**Giới thiệu**

ADO.NET là một bộ các thư viện hướng đối tượng (OOP) cho phép bạn tương tác với dữ liệu nguồn. Thông thường thì dữ liệu nguồn là một cơ sở dữ liệu (database), nhưng nó cũng có thể là file text, exel hoặc XML. Theo những mục tiêu của hướng dẫn này, chúng ta sẽ chỉ xem xét tới cách ADO.NET làm việc với database.

Như bạn có thể biết rằng, có rất nhiều loại database hiện nay như Microsoft SQL Server, Microsoft Access, Oracle, Borland Interbase, và IBM DB2,… Để làm rõ hơn phạm vi của loạt bài này, tất cả ví dụ sẽ sử dụng SQL Server.

**Data Provider**

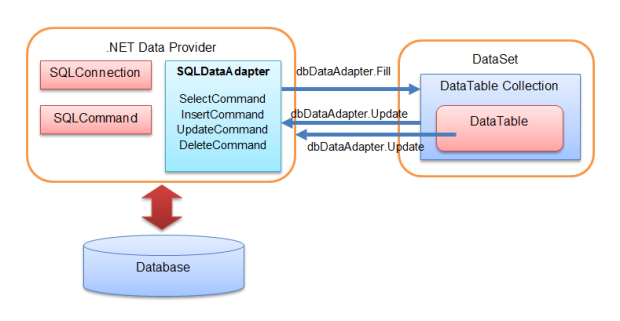
Chúng ta biết rằng ADO.NET cho phép tương tác với các loại dữ liệu và kiểu database. Mỗi loại dữ liệu cần một cách thức khác nhau để có thể truy xuất. Các loại dữ liệu cũ sử dụng giao thức ODBC, các loại dữ liệu mới hơn sử dụng giao thức OleDb. Vì vậy cần có một thư viện thống nhất để làm việc với chúng, đây chính là lý do mà ADO.NET được tạo ra.

ADO.NET cung cấp một cách thức chung để tương tác với nguồn dữ liệu, nhưng với mỗi loại dữ liệu bạn phải sử dụng một thư viện khác nhau. Các thư viện này được gọi là Data Provider và thường được đặt tên theo giao thức hoặc loại dữ liệu mà chúng cho phép bạn truy xuất. Table 1 liệt kê các data provider phổ biến, tiền tố (prefix) API mà chúng sử dụng và kiểu dữ liệu mà bạn có thể làm việc.

**table 1.  ADO.NET Data Providers are class libraries that allow a common way to interact with specific data sources or protocols.  The library APIs have prefixes that indicate which provider they support.**

|  |  |  |
| --- | --- | --- |
| **Provider Name** | **API prefix** | **Data Source Description** |
| ODBC Data Provider | Odbc | Data Sources with an ODBC interface.  Normally older data bases. |
| OleDb Data Provider | OleDb | Data Sources that expose an OleDb interface, i.e. Access or Excel. |
| Oracle Data Provider | Oracle | For Oracle Databases. |
| SQL Data Provider | Sql | For interacting with Microsoft SQL Server. |
| Borland Data Provider | Bdp | Generic access to many databases such as Interbase, SQL Server, IBM DB2, and Oracle. |

Một ví dụ có thể giúp bạn hiểu nghĩa của API prefix. Một trong những đối tượng ADO.NET bạn sẽ học là connection, chúng cho phép thiết lập một kết nối đến nguồn dữ liệu. Nếu bạn dùng OleDb Data Provider để kết nối, bạn sẽ cần dùng để đối tượng connection tên là OleDbConnection. Tương tự, đối tượng connection sẽ dùng prefix là Odbc hoặc Sql cho đối tượng OdbcConnection hoặc SqlConnection để kết nối với nguồn dữ liệu tương ứng. Như đã nói lúc đầu, ta sẽ học cách làm việc với SQL Server nên các đối tượng API của chúng ta sẽ có prefix là Sql.

[](https://yinyangit.wordpress.com/2011/07/31/ado-net-tutorial-lesson-01-introduction-to-ado-net/ado-net-data-access-architecture/)(Vẽ theo hình minh họa “**ADO.NET data access architecture”**từ MSDN: <http://msdn.microsoft.com/en-us/library/aa302163.aspx>)

**Các đối tượng của ADO.NET**

ADO.NET bao gồm nhiều đối tượng bạn có thể dùng với dữ liệu. Phần này giới thiệu một vài đối tượng chính bạn sẽ sử dụng. Theo tiến độ trong các bài viết sau, bạn sẽ khám phá thêm nhiều đối tượng ADO.NET và cách chúng được sử dụng trong mỗi lesson riêng. Các đối tượng dưới đây bắt buộc bạn phải biết để làm việc với dữ liệu khi dùng ADO.NET.

**SqlConnection**

Để tương tác với database, bạn phải có một kết nối tới nó. Kết nối giúp xác định database server, database name, user name, password, và các tham số cần thiết để kết nối tới database. Một đối tượng connection được dùng bởi đối tượng command vì thế chúng sẽ biết database nào để thực thi lệnh.

**SqlCommand**

Quá trình tương tác với database cần phải biết hành động nào bạn muốn xảy ra. Điều này được thực hiện bởi đối tượng command. Bạn dùng đối tượng command để gửi một câu lệnh SQL tới database. Một đối tượng command dùng một đối tượng connection để xác định database nào sẽ được truy xuất. Bạn có thể dùng một đối tượng command riêng lẻ để thực thi lệnh trực tiếp, hoặc để gắn một tham chiếu của đối tượng command cho một SqlDataAdapter – đối tượng giữ các command sẽ làm việc trên một nhóm dữ liệu như sẽ đề cập tới trong phần dưới.

**SqlDataReader**

Nhiều thao tác dữ liệu đòi hỏi bạn chỉ lấy một luồng dữ liệu để đọc. Đối tượng data reader cho phép bạn lấy được kết quả của một câu lệnh SELECT từ một đối tượng command. Để tăng hiệu suất, dữ liệu trả về từ một data reader là một luồng dữ liệu fast forward-only. Có nghĩa là bạn chỉ có thể lấy dữ liệu từ luồng theo một thứ tự nhất định. Mặc dù điều này có lợi về mặt tốc độ, nhưng nếu bạn cần phải thao tác dữ liệu, thì một DataSet sẽ là một đối tượng tốt hơn để làm việc.

**DataSet**

Đối tượng DataSet là một thể hiện của dữ liệu trong bộ nhớ. Chúng chứa nhiều đối tượng DataTable, bên trong DataTable lại có nhiều column và row, giống như các database table thông thường. Bạn thậm chí có thể định nghĩa dữ liệu giữa các table để tạo các quan hệ parent-child. DataSet được thiết kế đặc biệt để giúp quản lý dữ liệu trong bộ nhớ và để hỗ trợ các thao tác không cần kết nối (disconnected) trên dữ liệu. DataSet là một đối tượng được dùng bởi tất cả Data Provider, đó là lý do tại sao nó không có một Data Provider prefix trong tên gọi.

**SqlDataAdapter**

Đôi lúc dữ liệu mà bạn làm việc là read-only và bạn ít khi cần thay đổi dữ liệu nguồn. Vài trường hợp cần lưu trữ tạm dữ liệu trong bộ nhớ để hạn chế truy xuất đến database. Data adapter làm điều này dễ dàng bằng cách giúp bạn quản lý dữ liệu trong chế độ ngắt kết nối. Data adapter sẽ đổ vào DataSet khi đọc dữ liệu và thực hiện thay đổi dữ liệu một lượt vào database.

Data adapter chứa một tham chiếu đến đối tượng connection và mở/đóng kết nối tự động khi đọc và ghi dữ liệu vào database. Hơn nữa, data adapter chứa đối tượng command cho những thao tác SELECT, INSERT, UPDATE và DELETE trên dữ liệu. Bạn sẽ có một data adapter được định nghĩa cho mỗi table trong một DataSet và nó sẽ quản lý các giao tiếp với database cho bạn. Tất cả những gì bạn cần làm là chỉ cho data adapter khi nào nạp hoặc ghi vào database.

**Tổng kết**

ADO.NET là một kĩ thuật .NET để thao tác với nguồn dữ liệu. Bạn có một vài Data Provider, cho phép bạn giao tiếp với các nguồn dữ liệu khác nhau, dựa trên giao thức mà chúng dùng hoặc kiểu database. Không cần quan tâm đến điều này, với mỗi Data Provider được sử dụng, bạn sẽ dùng các đối tượng tương tự nhau để thao tác với dữ liệu. Đối tượng SqlConnection cho phép bạn quản lý một kết nối đến nguồn dữ liệu. SqlCommand cho phép bạn gửi lệnh đến dữ liệu. Để đọc dữ liệu nhanh theo cơ chế forward-only, sử dụng SqlDataReader. Nếu bạn muốn làm việc với dữ liệu đã ngắt kết nối, dùng một DataSet và hiện thực việc đọc và ghi đến dữ liệu nguồn bằng một SqlDataAdalter.

Nguồn tham khảo: <http://www.csharp-station.com/Tutorials/AdoDotNet/Lesson01.aspx>

# [[ADO.NET Tutorial] Lesson 02: Đối tượng SqlConnection](https://yinyangit.wordpress.com/2011/08/01/ado-net-tutorial-lesson-02-the-sqlconnection-object/)

[](https://yinyangit.wordpress.com/2011/08/01/ado-net-tutorial-lesson-02-the-sqlconnection-object/ado-net-database-connection-model/)**(The C# Station ADO.NET Tutorial)**

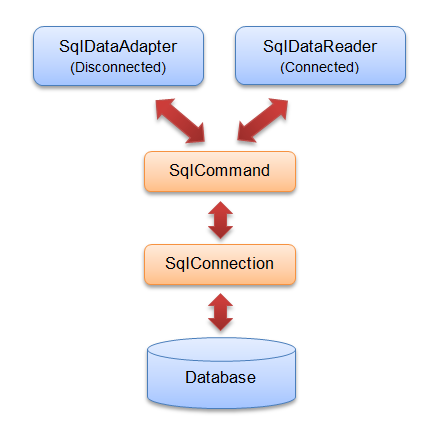
Lesson này sẽ mô tả về đối tượng SqlConnection và cách để kết nối tới database. Qua đó, bạn sẽ hiểu được các kiến thức về tạo, sử dụng và quản lý các đối tượng connection sao cho hiệu quả trong ADO.NET

## Giới thiệu

Điều đầu tiên bạn cần là tạo một kết nối để xác định database nào cần làm việc. Nó sẽ quản lý tất cả các logic ở mức thấp kết hợp với các giao thức đặc trưng của database. Công việc mà bạn cần làm chỉ là tạo một đối tượng connection, mở kết nối và đóng kết nối sau khi đã hoàn thành công việc.

Mặc dù làm việc với connection rất đơn giản trong ADO.NET, bạn vẫn cần hiểu về connection để có một quyết định đúng khi viết mã lệnh truy xuất dữ liệu. Phải hiểu rằng mỗi kết nối là một tài nguyên quan trọng. Nếu bạn chỉ có một ứng dụng làm việc với database trong máy, bạn có thể không cần quan tâm đến nó. Tuy nhiên, với các ứng dụng doanh nghiệp, hàng trăm người dùng khắp công ty kết nối tới cùng một database. Mỗi kết nối  giống như một đường đây và số lượng của chúng nằm trong mức giới hạn. Trường hợp bạn có thể thấy rõ hơn là  một web site được truy cập với hàng trăm nghìn lượt mỗi ngày. Ứng dụng sẽ bắt và giữ các kết nối, điều này có thể gây ra những ảnh hưởng nghiêm trọng đến hiệu suất và vận hành của ứng dụng.

Mô hình sau cho ta thấy cách mà SqlConnection được sử dụng bởi các đối tượng ADO.NET khác:

[](https://yinyangit.wordpress.com/2011/08/01/ado-net-tutorial-lesson-02-the-sqlconnection-object/ado-net-database-connection-model/)

## Tạo một đối tượng SqlConnection

Một đối tượng SqlConnection giống như các đối tượng khác trong C#. Bạn chỉ cần khai báo một thể hiện của SqlConnection, như dưới đây:

*SqlConnection conn = new SqlConnection(  
“Data Source=(local);Initial Catalog=Northwind;Integrated Security=SSPI”);*

Đối tượng SqlConnection trên sử dụng constructor với một tham số kiểu string. Tham số này được gọi là chuỗi kết nối (connection string). Table 1 mô tả các phần của một connection string.

###### **Table 1.  ADO.NET Connection String chứa các cặp key/value để xác định cách tạo một kết nối đến database.  Chúng bao gồm vị trí, tên của database và và chế độ bảo mật.**

|  |  |
| --- | --- |
| **Connection String Parameter Name** | **Description** |
| Data Source | Identifies the server.  Could be local machine, machine domain name, or IP Address. |
| Initial Catalog | Database name. |
| Integrated Security | Set to [SSPI](http://msdn.microsoft.com/en-us/library/aa380493(v=vs.85).aspx) to make connection with user’s Windows login |
| User ID | Name of user configured in SQL Server. |
| Password | Password matching SQL Server User ID. |

Integrated Security sẽ bảo mật khi bạn làm việc trên một máy đơn. Tuy nhiên, bạn sẽ thường xuyên cần phải định rõ mức bảo mật dựa trên SQL Server User ID với quyền hạn được xác định cho ứng dụng bạn sử dụng. Đoạn mã sau sử dụng một connection string với tham số User ID và Password:

*SqlConnection conn = new SqlConnection(  
“Data Source=DatabaseServer;Initial Catalog=Northwind;User ID=YourUserID;Password=YourPassword”);*

Lưu ý Data Source được gán cho DatabaseServer để chỉ ra rằng bạn có thể định danh một database trên một máy khác, thông qua mạng LAN, hoặc qua Internet. Ngoài ra, User ID và Password được thay thế cho tham số Integrated Security.

## Sử dụng SqlConnection

Mục đích của việc tạo một đối tượng SqlConnection là để các mã lệnh ADO.NET khác có thể làm việc được với database. Các đối tượng ADO.NET khác, như SqlCommand và SqlDataAdapter dùng một connection như một tham số. Quá trình sử dụng SqlConnection gồm các bước sau:

1. Tạo một SqlConnection.
2. Mở connection.
3. Truyền connection  cho các đối tượng ADO.NET khác.
4. Thực hiện các thao tác database với các đối tượng ADO.NET này.
5. Đóng connection.

Chúng ta đã thấy cách để tạo một đối tượng SqlConnection. Những bước còn lại, mở, truyền, sử dụng và đóng connection được thể hiện trong Listing 1.

##### Listing 1.  Using a SqlConnection

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54 | using System;  using System.Data;  using System.Data.SqlClient;    /// <summary>  /// Demonstrates how to work with SqlConnection objects  /// </summary>  class SqlConnectionDemo  {      static void Main()      {          // 1. Instantiate the connection          SqlConnection conn = new SqlConnection(              "Data Source=(local);Initial Catalog=Northwind;Integrated Security=SSPI");            SqlDataReader rdr = null;            try          {              // 2. Open the connection              conn.Open();                // 3. Pass the connection to a command object              SqlCommand cmd = new SqlCommand("select \* from Customers", conn);                //              // 4. Use the connection              //                // get query results              rdr = cmd.ExecuteReader();                // print the CustomerID of each record              while (rdr.Read())              {                  Console.WriteLine(rdr[0]);              }          }          finally          {              // close the reader              if (rdr != null)              {                  rdr.Close();              }                // 5. Close the connection              if (conn != null)              {                  conn.Close();              }          }      }  } |

Như đã thấy trong Listing 1, bạn mở một connection bằng cách gọi phương thức Open của đối tượng SqlConnection, **conn**. Bất kì thao tác nào thực hiện với SqlCommand cũng sử dụng connection này.

Mã lệnh sử dụng connection là một SqlCommand – đối tượng thực hiện truy vấn đến bảng Customers trong database. Tập kết quả trả về dưới dạng một SqlDataReader và vòng lặp while đọc cột đầu tiên của mỗi dòng trong tập kết quả này, chính là cột CustomerID. Chúng ta sẽ thảo luận về đối tượng SqlCommand và SqlDataReader trong lesson sau. Hiện tại, điều quan trọng là bạn cần hiểu rằng các đối tượng sử dụng SqlConnection để biết được database nào sẽ được truy xuất.

Khi bạn sử dụng xong đối tượng connection, bạn phải đóng nó. Thực hiện thất bại có thể gây ra các hậu quả nghiêm trọng đến hiệu suất và khả năng của ứng dụng. Như bạn thấy trong Listing 1: phương thức Close() được gọi trong một khối **finally** và chúng ta đảm bảo rằng connection sẽ khác null trước khi đóng nó.

*Lưu ý rằng chúng ta bao đoạn mã ADO.NET trong một khối*try/finally*. Khối*finally*giúp đảm bảo rằng một phần mã lệnh sẽ được thực thi, bất kể một ngoại lệ có xảy ra hay không. Bởi vì các connection là một tài nguyên hệ thống ít ỏi, bạn sẽ cần phải chắc chắn rằng chúng được đóng trong khổi*finally*.*

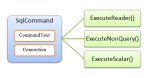
Một điểm bạn nên làm khi đóng connection là đảm bảo connection khác null. Nếu có sai sót xảy ra khi tạo connection, nó sẽ là null và bạn không nên cố đóng một connection không hợp lệ, điều này sẽ gây ra một ngoại lệ.

Ví dụ này cho thấy cách để dùng đối tượng SqlConnection với SqlDataReader – đối tượng yêu cầu bạn phải thực hiện một lệnh đóng connection rõ ràng. Tuy nhiên, khi dùng mô hình dữ liệu ngắt kết nối, bạn không cần phải mở và đóng connection thủ công. Chúng ta sẽ xem xét về vấn đề này trong lesson sau khi tìm hiểu về đối tượng SqlDataAdapter.

## Tổng kết

Các đối tượng SqlConnection cho phép các đối tượng ADO.NET khác biết database nào sẽ được truy xuất và cách để tạo một connection. Chúng được tạo bằng cách truyền một connection string với một danh sách các cặp key/value để định nghĩa connection. Các bước mà bạn quản lý một connection là tạo, mở, truyền (cho các đối tượng ADO.NET khác), sử dụng và đóng. Hãy đảm bảo đóng connection một cách chính xác khi bạn đã làm việc xong với nó để chắc chắn rằng bạn không có một sự rò rỉ tài nguyên kết nối.

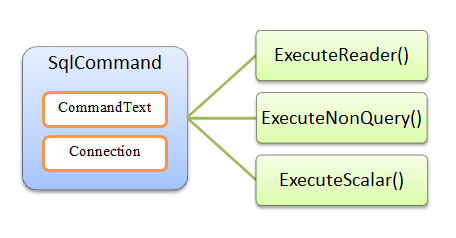
# [[ADO.NET Tutorial] Lesson 03: Đối tượng SqlCommand](https://yinyangit.wordpress.com/2011/08/02/ado-net-tutorial-lesson-03-the-sqlcommand-object/)

**[](https://yinyangit.wordpress.com/2011/08/02/ado-net-tutorial-lesson-03-the-sqlcommand-object/ado-net-sqlcommand-object/)(The C# Station ADO.NET Tutorial)**

Lesson này giới thiệu về đối tượng SqlCommand và cách sử dụng để thao tác với database. Qua bài viết này, bạn sẽ biết cách sử dụng các phương thức ExecuteReader(), ExecuteNonQuery() và ExecuteScalar() của đối tượng command.

## Giới thiệu

Đối tượng SqlCommand cho phép bạn chọn kiểu tương tác mà bạn muốn thực hiện với database. Ví dụ, bạn có thể thực hiện các lệnh select, insert, modify, và delete các dòng trong một table của database. Đối tượng này có thể được dùng để hỗ trợ mô hình quản lý dữ liệu ngắt kết nối (disconnected), nhưng trong bài học này chúng ta chỉ dùng đối tượng SqlCommand làm việc độc lập. Bài học này cũng cho bạn thấy cách để lấy một giá trị đơn từ database, như là số dòng của một table.

[](https://yinyangit.wordpress.com/2011/08/02/ado-net-tutorial-lesson-03-the-sqlcommand-object/ado-net-sqlcommand-object/)

## Tạo một đối tượng SqlCommand

Tương tự như các đối tượng C# khác, bạn tạo đối tượng SqlCommand bằng cách khai báo một thể hiện của nó:

*SqlCommand cmd = new SqlCommand(“select CategoryName from Categories”, conn);*

Dòng trên là điển hình để tạo một đối tượng SqlCommand. Nó lấy một tham số là chuỗi lệnh bạn cần thực thi và một tham chiếu đến đối tượng SqlConnection. SqlCommand có một vài overload, mà bạn sẽ thấy trong các ví dụ của tutorial này.

## Querying Data – Truy vấn dữ liệu

Khi dùng một lệnh SQL select, bạn lấy được một dữ liệu từ database để hiển thị. Để làm được điều này với SqlCommand, bạn cần dùng phương thức ExecuteReader() để trả về một đối tượng SqlDataReader. Chúng ta sẽ thảo luận về SqlDataReader trong bài tiếp theo. Ví dụ sau cho thấy cách dùng SqlCommand để lấy được một đối tượng SqlDataReader:

|  |  |
| --- | --- |
| 1  2  3  4  5 | // 1. Instantiate a new command with a query and connection  SqlCommand cmd = new SqlCommand("select CategoryName from Categories", conn);    // 2. Call Execute reader to get query results  SqlDataReader rdr = cmd.ExecuteReader(); |

Trong ví dụ trên, chúng ta tạo một đối tượng SqlCommand, truyền chuỗi lệnh và đối tượng connection vào constructor. Sau đó chúng ta lấy về đối tượng SqlDataReader bằng cách gọi phương thức ExecuteReader() của đối tượng SqlCommand, cmd.

## Inserting Data – Chèn dữ liệu

Để chèn dữ liệu vào database, dùng phương thức ExecuteNonQuery() của đối tượng SqlCommand. Đoạn code sau cho thấy cách chèn dữ liệu vào một bảng trong database:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | // prepare command string  string insertString = @"      insert into Categories      (CategoryName, Description)      values ('Miscellaneous', 'Whatever doesn''t fit elsewhere')";    // 1. Instantiate a new command with a query and connection  SqlCommand cmd = new SqlCommand(insertString, conn);    // 2. Call ExecuteNonQuery to send command  cmd.ExecuteNonQuery(); |

Lưu ý hai dấu nháy đơn (‘’) trong chuỗi insertString của từ “doesn’’t” (không phải dấu nháy kép). Đây là cách để bạn chèn một dấu nháy đơn vào chuỗi trong trường hợp chuỗi đó được bao bởi hai dấu nháy đơn (‘) thay vì nháy kép (“). SQL sẽ xem hai dấu nháy đơn đứng sát nhau là một dấu nháy đơn, tương tự như cách viết (\’) trong C#.

Một điểm cần chú ý nữa là chúng ta xác định rõ các cột CategoryName và Description. Bảng Categories có một primary key là CategoryID. Chúng ta không cần chèn dữ liệu vào cột này bởi vì SQL Server sẽ tự động thêm vào. Nếu bạn thử thêm giá trị vào cột primary key, như CategoryID, một ngoại lệ sẽ được tạo ra.

Để thực thi lệnh này, chỉ cần đơn giản gọi phương thức ExecuteNonQuery() của đối tượng SqlCommand, cmd.

## Updating Data – Cập nhật dữ liệu

Phương thức ExecuteNonQuery cũng được dùng để cập nhật dữ liệu, như đoạn mã sau:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | // prepare command string   string updateString = @"       update Categories       set CategoryName = 'Other'       where CategoryName = 'Miscellaneous'";     // 1. Instantiate a new command with command text only   SqlCommand cmd = new SqlCommand(updateString);     // 2. Set the Connection property   cmd.Connection = conn;     // 3. Call ExecuteNonQuery to send command   cmd.ExecuteNonQuery(); |

Một lần nữa, chúng ta đặt câu SQL trong một biến string, nhưng lần này chúng ta dùng một constructor khác của SqlCommand, chỉ có một tham số là câu lệnh SQL. Trong bước 2, chúng ta gán đối tượng SqlConnection, conn, cho property Connection của đối tượng SqlCommand, cmd.

Thay vì cách này, bạn có thể sử dụng constructor với hai tham số tương tự như cho đoạn mã insert phần trên. Nó cho thấy rằng bạn có thể thay đổi đối tượng connection gán cho một command bất kì lúc nào.

Bước cuối cùng, phương thức ExecuteNonQuery() thực thi lệnh update.

## Deleting Data – Xóa dữ liệu

Bạn cũng có thể xóa dữ liệu bằng phương thức ExecuteNonQuery(). Ví dụ sau cho thấy cách xóa một dòng từ database với phương thức ExecuteNonQuery():

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | // prepare command string   string deleteString = @"       delete from Categories       where CategoryName = 'Other'";     // 1. Instantiate a new command   SqlCommand cmd = new SqlCommand();     // 2. Set the CommandText property   cmd.CommandText = deleteString;     // 3. Set the Connection property   cmd.Connection = conn;     // 4. Call ExecuteNonQuery to send command   cmd.ExecuteNonQuery(); |

Ví dụ này sử dụng constructor không có tham số của SqlCommand. Thay vào đó, nó gán cho hai property CommandText và Connection của đối tượng SqlCommand, cmd.

Chúng ta cũng có thể dùng hai overload constructor trước đó của SqlCommand, dùng để insert và update, với kết quả tương tự. Điều này cho thấy rằng bạn có thể thay đôi cả chuỗi lệnh và đối tượng connection bất kì lúc nào.

Lời gọi phương thức ExecuteNonQuery()  gửi lệnh đến database.

## Lấy một giá trị đơn

Đôi lúc tất cả những gì bạn cần từ database chỉ là một giá trị đơn, đó có thể là một giá trị đếm, tổng, trung bình, hoặc các giá trị kết hợp khác từ dữ liệu. Thực thi phương thức ExecuteReader() để lấy về một đối tượng SqlDataReader và tính toán kết quả trong mã lệnh của bạn không phải cách tốt để làm điều này. Cách tốt nhất là để database làm công việc này và trả về giá trị bạn cần. Ví dụ sau cho thấy cách làm điều này với phương thức ExecuteScalar():

|  |  |
| --- | --- |
| 1  2  3  4  5 | // 1. Instantiate a new command  SqlCommand cmd = new SqlCommand("select count(\*) from Categories", conn);    // 2. Call ExecuteNonQuery to send command  int count = (int)cmd.ExecuteScalar(); |

Câu truy vấn trong constructor của  SqlCommand sẽ lấy về giá trị đếm của tất cả dòng trong bảng Categories. Câu truy vấn này chỉ trả về một giá trị đơn. Phương thức ExecuteScalar() trong bước 2 trả về giá trị này. Bởi vì kiểu trả về của ExecuteScalar() là object, chúng ta cần ép kiểu để chuyển giá trị sang int.

## Kết hợp tất cả với nhau

Để đơn giản, tôi cho thấy các đoạn mã ngắn trong những phần trước để minh họa các kĩ thuật tương ứng. Nó cũng rất hữu ích để xem cách mà mã lệnh thực hiện trong một đoạn mã hoàn chỉnh của chương trình. Listing 1 cho thấy tất cả mã lệnh được dùng trong ví dụ này, cùng với phương thức Main() để thực thi và hiển thị kết quả xuất ra:

##### Listing 1.  SqlConnection Demo

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126  127  128  129  130  131  132  133  134  135  136  137  138  139  140  141  142  143  144  145  146  147  148  149  150  151  152  153  154  155  156  157  158  159  160  161  162  163  164  165  166  167  168  169  170  171  172  173  174  175  176  177  178  179  180  181  182  183  184  185  186  187  188  189  190  191  192  193  194  195  196  197  198  199  200  201  202  203  204  205  206  207  208  209  210  211  212  213  214  215  216  217  218  219  220  221  222  223  224  225  226  227  228  229  230  231  232  233  234  235  236  237 | using System;   using System.Data;   using System.Data.SqlClient;     ///  <summary> /// Demonstrates how to work with SqlCommand objects   /// </summary>   class SqlCommandDemo   {       SqlConnection conn;         public SqlCommandDemo()       {           // Instantiate the connection           conn = new SqlConnection(              "Data Source=(local);Initial Catalog=Northwind;Integrated Security=SSPI");       }         // call methods that demo SqlCommand capabilities       static void Main()       {           SqlCommandDemo scd = new SqlCommandDemo();             Console.WriteLine();           Console.WriteLine("Categories Before Insert");           Console.WriteLine("------------------------");             // use ExecuteReader method           scd.ReadData();             // use ExecuteNonQuery method for Insert           scd.Insertdata();           Console.WriteLine();           Console.WriteLine("Categories After Insert");           Console.WriteLine("------------------------------");            scd.ReadData();             // use ExecuteNonQuery method for Update           scd.UpdateData();             Console.WriteLine();           Console.WriteLine("Categories After Update");           Console.WriteLine("------------------------------");             scd.ReadData();             // use ExecuteNonQuery method for Delete           scd.DeleteData();             Console.WriteLine();           Console.WriteLine("Categories After Delete");           Console.WriteLine("------------------------------");             scd.ReadData();             // use ExecuteScalar method           int numberOfRecords = scd.GetNumberOfRecords();             Console.WriteLine();           Console.WriteLine("Number of Records: {0}", numberOfRecords);       }         ///  <summary> /// use ExecuteReader method   /// </summary>       public void ReadData()       {          SqlDataReader rdr = null;             try           {               // Open the connection               conn.Open();                 // 1. Instantiate a new command with a query and connection               SqlCommand cmd = new SqlCommand("select CategoryName from Categories", conn);                 // 2. Call Execute reader to get query results               rdr = cmd.ExecuteReader();                 // print the CategoryName of each record               while (rdr.Read())               {                   Console.WriteLine(rdr[0]);               }           }           finally           {               // close the reader               if (rdr != null)               {                   rdr.Close();               }                 // Close the connection               if (conn != null)               {                   conn.Close();               }           }       }         ///  <summary> /// use ExecuteNonQuery method for Insert   /// </summary>       public void Insertdata()       {           try           {               // Open the connection               conn.Open();                 // prepare command string               string insertString = @"                   insert into Categories                   (CategoryName, Description)                   values ('Miscellaneous', 'Whatever doesn''t fit elsewhere')";                 // 1. Instantiate a new command with a query and connection               SqlCommand cmd = new SqlCommand(insertString, conn);                 // 2. Call ExecuteNonQuery to send command               cmd.ExecuteNonQuery();           }           finally           {               // Close the connection               if (conn != null)               {                   conn.Close();               }           }       }         ///  <summary> /// use ExecuteNonQuery method for Update   /// </summary>       public void UpdateData()       {           try           {               // Open the connection               conn.Open();                 // prepare command string               string updateString = @"                   update Categories                   set CategoryName = 'Other'                   where CategoryName = 'Miscellaneous'";                 // 1. Instantiate a new command with command text only               SqlCommand cmd = new SqlCommand(updateString);                 // 2. Set the Connection property               cmd.Connection = conn;                 // 3. Call ExecuteNonQuery to send command               cmd.ExecuteNonQuery();          }           finally           {               // Close the connection               if (conn != null)               {                   conn.Close();               }           }       }         ///  <summary> /// use ExecuteNonQuery method for Delete   /// </summary>       public void DeleteData()       {           try           {               // Open the connection               conn.Open();                 // prepare command string               string deleteString = @"                   delete from Categories                   where CategoryName = 'Other'";                 // 1. Instantiate a new command               SqlCommand cmd = new SqlCommand();                 // 2. Set the CommandText property               cmd.CommandText = deleteString;                 // 3. Set the Connection property               cmd.Connection = conn;                 // 4. Call ExecuteNonQuery to send command               cmd.ExecuteNonQuery();           }           finally           {               // Close the connection               if (conn != null)               {                   conn.Close();               }           }       }         ///  <summary> /// use ExecuteScalar method   /// </summary>       /// number of records       public int GetNumberOfRecords()       {           int count = -1;             try           {               // Open the connection               conn.Open();                 // 1. Instantiate a new command               SqlCommand cmd = new SqlCommand("select count(\*) from Categories", conn);                 // 2. Call ExecuteScalar to send command               count = (int)cmd.ExecuteScalar();           }           finally           {              // Close the connection               if (conn != null)               {                   conn.Close();               }           }           return count;       }   } |

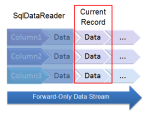
Trong Lising 1, đối tượng SqlConnection được khởi tạo trong lớp SqlCommandDemo. Điều này hợp lệ vì đối tượng sẽ được dọn dẹp khi CLR garbage collector làm việc. Điều quan trọng là ta đóng connection sau khi sử dụng nó. Chương trình mở connection trong một khổi try và đóng nó trong một khối finally cho mỗi phương thức.

Phương thức ReadData() hiển thị nội dung của cột CategoryName của bảng Categories. Chúng ta dùng nó vài lần trong phương thức Main() để hiển thị trạng thái hiện tại của bảng Categories. Bảng này sẽ bị thay đổi khi mỗi lệnh insert, update, delete được thực hiện.

## Tổng kết

Đối tượng SqlCommand cho phép bạn truy vấn và gửi lệnh đến một database. Nó có các phương thức sử dụng cho các lệnh khác nhau. Phương thức ExecuteReader() trả về một đối tượng SqlDataReader để hiển thị kết quả của câu truy vấn. Cho các lệnh insert, update và delete, bạn dùng phương thức ExecuteNonQuery(). Nếu bạn chỉ cần một giá trị đơn từ một câu truy vấn, phương thức ExecuteScalar() là lựa chọn tốt nhất.

# [[ADO.NET Tutorial] Lesson 04: Đọc dữ liệu với SqlDataReader](https://yinyangit.wordpress.com/2011/08/03/ado-net-tutorial-lesson-04-reading-data-with-sqldatareader/)

**[](https://yinyangit.wordpress.com/2011/08/03/ado-net-tutorial-lesson-04-reading-data-with-sqldatareader/ado-net-sqldatareader-forward-only-data-stream/)(The C# Station ADO.NET Tutorial)**

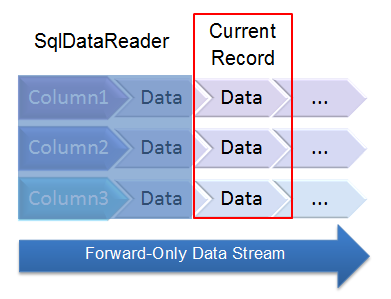
Lesson này hướng dẫn các dùng SqlDataReader để đọc dữ liệu, qua đó bạn sẽ biết được cách dùng SqlDataReader cho những mục đích phù hợp.

## Giới thiệu

Một SqlDataReader là đối tượng phù hợp để đọc dữ liệu một cách hiệu quả nhất. Như tên gọi, bạn không thể dùng nó để ghi dữ liệu. SqlDataReader thường được mô tả là luồng dữ liệu fast-forward firehose-like (fire-hose: ống vòi rồng).

Bạn có thể đọc dữ liệu từ đối tượng SqlDataReader theo hướng forward-only trong một thứ tự nhất định. Mỗi lần đọc một vài dữ liệu, bạn phải lưu nó nếu cần thiết bởi vì bạn không thể quay trở lại và đọc nó một lần nữa.

Kiểu thiết kế forward-only của SqlDataReader để giúp nó hoạt động nhanh. Nó không thể di chuyển trực tiếp đến các dòng dữ liệu ở vị trí bất kì và không thể ghi vào dữ liệu nguồn. Do đó, nếu bạn chỉ yêu cầu đọc một nhóm dữ liệu một lần và cần phương pháp nhanh nhất, SqlDataReader là lựa chọn tốt nhất.

[](https://yinyangit.wordpress.com/2011/08/03/ado-net-tutorial-lesson-04-reading-data-with-sqldatareader/ado-net-sqldatareader-forward-only-data-stream/)

**Note**: Tôi dùng từ “một lần” trong đoạn trước khi nói về lý do tại sao bạn nên dùng một SqlDataReader. Tuy nhiên có những ngoại lệ và trường hợp, sẽ hiệu quả hơn nếu bạn dùng một DataSet để lưu dữ liệu tạm. Vấn đề này nằm ngoài phạm vi của bài này, chúng ta sẽ thảo luận về đối tượng DataSet trong bài tiếp theo.

## Tạo một đối tượng SqlDataReader

Có một chút khác biệt để lấy được một thể hiện của SqlDataReader so với các đối tượng ADO.NET khác. Bạn phải gọi phương thức ExecuteReader() của một đối tượng SqlCommand, như:

*SqlDataReader rdr = cmd.ExecuteReader();*

Phương thức ExecuteReader() của đối tượng SqlCommand, cmd, trả về một thể hiện của SqlDataReader. Tạo một SqlDataReader với toán tử **new**không có tác dụng gì cho bạn cả. Như bạn đã học trong bài trước, đối tượng SqlCommand chứa tham chiếu đến connection và câu lệnh SQL cần thiết để SqlDataReader lấy được dữ liệu.

## Đọc dữ liệu

Bài trước chứa đoạn mã sử dụng SqlDataReader, nhưng chưa được thảo luận tới vì chúng ta sẽ tập trung vào một chủ đề riêng. Bài này được tạo ra từ những gì bạn đã xem và giải thích cách sử dụng SqlDataReader.

Như đã giải thích tước, SqlDataReader trả về dữ liệu qua một luồng liên tục. Để đọc dữ  liệu, bạn phải lấy dữ liệu từ một bảng từng dòng một (row-by-row). Mỗi lần một dòng được đọc, dòng trước đó sẽ không còn hiệu lực. Để đọc lại dòng đó, bạn cần phải tạo một thể hiện mới của SqlDataReader và đọc xuyên qua luồng dữ liệu một lần nữa.

Phương pháp điển hình để đọc từ luồng dữ liệu trả về bởi SqlDataReader là lặp qua mỗi dòng với một vòng lặp while. Đoạn code sau cho thấy cách làm điều này:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | while (rdr.Read())  {      // get the results of each column      string contact = (string)rdr["ContactName"];      string company = (string)rdr["CompanyName"];      string city    = (string)rdr["City"];        // print out the results      Console.Write("{0,-25}", contact);      Console.Write("{0,-20}", city);      Console.Write("{0,-25}", company);      Console.WriteLine();  } |

Lưu ý phương thức Read() của SqlDataReader, rdr, trong điều kiện của vòng lặp while trong đoạn code trên. Giá trị trả về của Read() là kiểu bool và trả về **true** đến khi nào vẫn còn dòng để đọc. Sau khi dòng cuối được đọc trong luồng dữ liệu, Read() trả về **false**.

Trong bài trước, chúng ta lấy cột đầu tiên của dòng bằng cách dùng indexer của SqlDataReader, ví dụ rdr[0]. Bạn có thể lấy mỗi cột của dòng với indexer kiểu số, nhưng cách này rất khó đọc. Ví dụ trên dùng indexer kiểu string, với chuỗi sử dụng là tên cột từ câu truy vân SQL (tên này sẽ trùng với tên cột của bảng nếu bạn dùng dấu sao \* trong lấy dữ liệu). Indexer kiểu string sẽ dễ đọc hơn và giúp mã nguồn dễ bảo trì.

Bỏ qua kiểu tham số của indexer, một indexer của SqlDataReader sẽ trả về kiểu object. Đó là lý do tại sao ví dụ trên phải chuyển kết quả về kiểu string. Mỗi lần giá trị được lấy ra, bạn có thể làm bất kì điều gì bạn muốn với chúng, như in ra màn hình với các phương thức của lớp Console.

Bổ sung: Tuy nhiên thay vì sử dụng indexer của SqlDataReader, bạn có thể dùng các phương thức GetXXX() với XXX là kiểu dữ liệu bạn muốn lấy. Tùy vào tên phương thức mà kiểu trả về sẽ tương ứng với nó. Ví dụ cột đầu tiên của bảng là string, bạn có thể gọi như sau:

*string value = rdr.GetString(0);*

Tuy nhiên như bạn thấy, tham  số của nó là kiểu int, một số chỉ vị trí của cột tính từ 0.

Bạn có thể xem chi tiết hơn tại: <http://msdn.microsoft.com/en-us/library/system.data.sqlclient.sqldatareader.aspx>.

Bạn có thể giải quyết điều này bằng cách dùng phương thức GetOrdinal(string name) với tham số là tên cột và giá trị trả về là vị trí của cột. Như vậy bạn có viết một phương thức mới (trong trường hợp bạn muốn một tạo sub class hay wrapper của SqlDataReader) chấp nhận tham số là tên cột thay vì vị trí, một minh họa đơn giản:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | string GetString(string columnName)  {      int columnIndex = rdr.GetOrdinal(columnName);      string value =rdr.GetString(columnIndex);      return value;  } |

## Hoàn tất

Luôn nhớ việc đóng SqlDataReader, giống như bạn cần phải đóng SqlConnection. Cho đoạn mã truy xuất dữ liệu trong một khối try và đặt lệnh đóng trong khối finally, giống như sau:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | try  {      // data access code  }  finally  {      // 3. close the reader      if (rdr != null)      {          rdr.Close();      }        // close the connection too  } |

Đoạn mã trên kiểm tra SqlDataReader để đảm bảo rằng nó khác null. Sau khi đã xác nhận đó là một thể hiện hợp lệ của SqlDataReader, nó sẽ được đóng lại. Listing 1 cho thấy đoạn mã hoàn chỉnh mà bạn đã thấy một vài phần của nó ở trên:

##### Listing 1: Using the SqlDataReader

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71 | using System;  using System.Data;  using System.Data.SqlClient;    namespace Lesson04  {      class ReaderDemo      {          static void Main()          {              ReaderDemo rd = new ReaderDemo();              rd.SimpleRead();          }            public void SimpleRead()          {              // declare the SqlDataReader, which is used in              // both the try block and the finally block              SqlDataReader rdr = null;                // create a connection object              SqlConnection conn = new SqlConnection("Data Source=(local);Initial Catalog=Northwind;Integrated Security=SSPI");                // create a command object              SqlCommand cmd  = new SqlCommand(                  "select \* from Customers", conn);                try              {                  // open the connection                  conn.Open();                    // 1.  get an instance of the SqlDataReader                  rdr = cmd.ExecuteReader();                    // print a set of column headers                  Console.WriteLine("Contact Name             City                Company Name");                  Console.WriteLine("------------             ------------        ------------");                    // 2.  print necessary columns of each record                  while (rdr.Read())                  {                      // get the results of each column                      string contact = (string)rdr["ContactName"];                      string company = (string)rdr["CompanyName"];                      string city    = (string)rdr["City"];                        // print out the results                      Console.Write("{0,-25}", contact);                      Console.Write("{0,-20}", city);                      Console.Write("{0,-25}", company);                      Console.WriteLine();                  }              }              finally              {                  // 3. close the reader                  if (rdr != null)                  {                      rdr.Close();                  }                    // close the connection                  if (conn != null)                  {                      conn.Close();                  }              }          }      }  } |

## Tổng kết

Đối tượng SqlDataReader cho phép bạn đọc nhanh dữ liệu theo cách forward-only. Bạn lấy dữ liệu bằng cách đọc từng dòng từ luồng dữ liệu. Gọi phương thức Close() của SqlDataReader để đảm bảo sẽ không có tài nguyên nào thất thoát.

# [[ADO.NET Tutorial] Lesson 05: Làm việc với Disconnected Data – DataSet và SqlDataAdapter](https://yinyangit.wordpress.com/2011/08/04/ado-net-tutorial-lesson-05-working-with-disconnected-data%e2%80%93dataset-and-sqldataadapter/)

**[](https://yinyangit.wordpress.com/2011/08/04/ado-net-tutorial-lesson-05-working-with-disconnected-data%e2%80%93dataset-and-sqldataadapter/ado-net-dataset-dataadapter/)(The C# Station ADO.NET Tutorial)**

Lesson này hướng dẫn cách làm việc với dữ liệu ngắt kết nối (disconnected data) với DataSet và SqlDataAdapter. Qua đó, bạn sẽ biết cách sử dụng DataSet, SqlDataAdapter  và hiểu tại sao cần phải làm việc với disconnected data.

## Giới thiệu

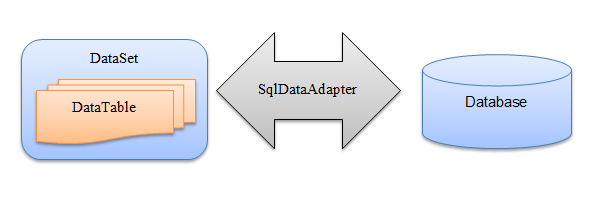
Một DataSet là một đối tượng chứa dữ liệu trong bộ nhớ và có thể gồm nhiều bảng. DataSet chỉ chứa dữ liệu chứ không tương tác với nguồn dữ liệu. Thay vào đó, SqlDataAdapter sẽ được dùng để quản lý các kết nối với nguồn dữ liệu và cho chúng ta chế độ làm việc disconnected. SqlDataAdapter mở  một kết nối chỉ khi cần thiết và đóng nó ngay sau khi tác vụ được hoàn thành. Ví dụ, SqlDataAdapter thực hiện các tác vụ sau, khi đổ dữ liệu vào DataSet:

1. Mở kết nối
2. Đổ dữ liệu vào DataSet (Fill)
3. Đóng kết nối

Và thực hiện các công việc sau, khi cập nhật dữ liệu nguồn với thay đổi của DataSet:

1. Mở kết nối
2. Ghi thay đổi từ DataSet vào dữ liệu nguồn (Update)
3. Đóng kết nối

Giữa hai thao tác Fill và Update, các kết nối với nguồn dữ liệu được đóng lại và bạn có thể tự do ghi, đọc dữ liệu với DataSet. Đây chính là cơ chế của mô hình làm việc với disconnected data. Bởi vì ứng dụng sẵn sàng kết nối khi cần thiết, ứng dụng trở nên dễ phát triển hơn.

[](https://yinyangit.wordpress.com/2011/08/04/ado-net-tutorial-lesson-05-working-with-disconnected-data%e2%80%93dataset-and-sqldataadapter/ado-net-dataset-dataadapter/)

Hai kịch bản minh họa cho lý do tại sao bạn cần làm việc với disconnected data: người dùng làm việc không cần kết nối mạng và giúp Web site dễ phát triển hơn. Hãy xem xét việc nhà kinh doanh cần dữ liệu của khách hàng trong khi họ đi công tác. Khi bắt đầu ngày, họ sẽ cần đồng bộ dữ liệu với database chính  để lấy các thông tin cập nhật mới nhất. Trong suốt ngày hôm đó, họ sẽ thực hiện các thay đổi trên dữ liệu khách hàng hiện tại, thêm khách hàng mới, và nhập các hóa đơn mới. Điều này hợp lệ vì họ có nguồn dữ liệu khách hàng riêng và những người khác không thể thay đổi trên cùng dòng dữ liệu. Vào cuối ngày, nhà kinh doanh cập nhật sẽ kết nối vào mạng và cập nhật thay đổi cho tiến trình xử lý ban đêm.

Kịch bản khác là giúp Web site dễ phát triển hơn.Với một SqlDataReader, bạn phải trở lại database lấy dữ liệu mỗi khi xem một trang. Điều này yêu cầu một kết nối mới cho mỗi lần tải trang, nó sẽ ảnh hưởng lớn đến hiệu suất khi số lượng người dùng tăng lên. Một cách để khắc phục điều này là dùng DataSet, chỉ cần cập nhật một lần và lưu trong bộ nhớ tạm (cache). Mỗi yêu cầu tải trang sẽ kiểm tra cache và nạp dữ liệu (từ database) nếu nó không tồn tại hoặc lấy dữ liệu ra khỏi cache và hiển thị nó. Điều này giúp hạn chế truy xuất database và tăng hiệu suất cho ứng dụng của bạn.

Các trường hợp ngoại lệ cho kịch bản trên bao gồm các trường hợp bạn cần cập nhật dữ liệu. Bạn phải quyết định, dựa trên cách dữ liệu sẽ được dùng và chiến lược của bạn. Dùng disconnected data khi thông tin của bạn thường là read-only, nhưng hãy xem xét các giải pháp thay thế (như dùng đối tượng SqlCommand để cập nhật tức thời) khi bạn yêu cầu một vài thứ linh hoạt hơn. Cũng vậy, nếu số lượng dữ liệu quá lớn để lưu trong bộ nhớ, bạn sẽ cần dùng SqlDataReader để đọc dữ liệu.

## Tạo đối tượng DataSet

Không có gì đặc biệt khi tạo một DataSet. Bạn chỉ cần tạo một thể hiện mới, giống bất kì đối tượng nào:

*DataSet dsCustomers = new DataSet();*

Constructor của DataSet không yêu cầu tham số. Tuy nhiên có một overload chấp nhận một chuỗi đại diện cho tên của DataSet, được dùng nếu bạn cần [serialize](https://yinyangit.wordpress.com/2011/06/02/net-serialization-and-deserialization-in-csharp/) dữ liệu thành XML. Bây giờ, ta có một DataSet rỗng và cần một SqlDataAdapter để nạp dữ liệu cho nó.

## Tạo một SqlDataAdapter

SqlDataAdapter chứa các lệnh SQL và đối tượng connection để đọc và ghi dữ liệu. Bạn khởi tạo nó với câu SQL select và đối tượng connection:

*SqlDataAdapter daCustomers = new SqlDataAdapter(“select CustomerID, CompanyName from Customers”, conn);*

Dòng mã trên tạo một đối tượng SqlDataAdapter, daCustomers. Câu SQL select xác định dữ liệu nào sẽ được đọc vào DataSet. Đối tượng connection, conn, nên được khởi tạo từ trước, nhưng không được mở. Đó là công việc của SqlDataAdapter để mở và đóng connection khi phương thức Fill() và Update() được gọi.

SqlDataAdapter tất cả lệnh cần thiết để tương tác với dữ liệu nguồn. Dòng mã trên xác định câu lệnh select, nhưng không cho thấy câu lệnh insert, update và delete. Chúng sẽ được thêm vào SqlDataAdapter sau khi nó được khởi tạo.

Có hai cách để thêm các lệnh insert, update, delete: thông qua các property của SqlDataAdapter hoặc với một SqlCommandBuilder. Trong bài này, tôi sẽ cho bạn thấy cách dễ nhất để làm điều này với SqlCommandBuilder. Trong bài sau, tôi sẽ sử dụng các property của SqlDataAdapter, công việc này sẽ đòi hỏi nhiều bước hơn nhưng sẽ giúp bạn hiệu quả hơn cách làm việc của SqlCommandBuilder. Đây là cách để thêm các lệnh vào SqlDataAdapter với SqlCommandBuilder.

*SqlCommandBuilder cmdBldr = new SqlCommandBuilder(daCustomers);*

Lưu ý rằng  dòng mã trên khởi tạo một đối tượng SqlCommandBuilder với constructor cần một tham số là SqlDataAdapter, daCustomers. Điều này giúp SqlCommandBuilder biết đối tượng SqlDataAdapter nào để thêm các lệnh vào. SqlCommandBuilder sẽ đọc câu SQL select (lấy từ SqlDataAdapter), từ đó suy ra các lệnh insert, update và delete, sau đó gán các lệnh mới vào các property Insert, Update, Delete của SqlDataAdapter tương ứng.

Như tôi đề cập từ trước, SqlCommandBuilder có hạn chế. Nó làm việc khi bạn làm một câu select đơn giản trên một bảng. Tuy nhiên, khi bạn cần kết hợp hai hoặc nhiều bảng hoặc thực thi một stored procedure, nó sẽ không làm việc.

## Đổ dữ liệu vào DataSet

Để đổ dữ liệu vào DataSet bạn cần dùng phương thức Fill() của SqlDataAdapter, như sau:

*daCustomers.Fill(dsCustomers, “Customers”);*

Phương thức Fill(), trong dòng trên lấy hai tham số: một DataSet và một tên bảng.  DataSet phải được tạo trước khi bạn đổ dữ liệu vào nó. Tham số thứ hai là tên của bảng sẽ được tạo trong DataSet. Bạn có thể đặt bất kì tên gì cho bảng. Thông thường, tôi sẽ để tên bảng trùng với tên gốc của nó trong database. Tuy nhiên, nếu câu select của SqlDataAdapter chứa một lệnh join, bạn sẽ cần phải đặt một tên rõ ràng khác cho bảng.

Phương thức Fill() có một overload chấp nhận một tham số là DataSet. Trong trường hợp này, bảng được tạo sẽ có tên mặc định là “table1” cho bảng đầu tiên. Số này sẽ tăng dần (table2, tabl3,…,tableN) cho mỗi bảng thêm vào DataSet nếu như tên bảng không được chỉ ra trong phương thức Fill().

## Sử dụng DataSet

Một DataSet sẽ gắn dữ liệu vào DataGrid của ASP.NET và Windows form. Đây là một ví dụ sẽ gán DataSet cho một Windows forms DataGrid:

*dgCustomers.DataSource = dsCustomers;  
dgCustomers.DataMember = “Customers”;*

Điều đầu tiên chúng ta làm, trong đoạn mã trên, là gán DataSet cho property DataSource của DataGrid. Điều này giúp DataGrid biết nó có dữ liệu được gắn vào, tuy nhiên bạn sẽ thấy một dấu ‘+’ trên GUI bởi vì DataSet có thể giữ nhiều bảng và DataGrid cho phép bạn mở rộng ra để xem mỗi bảng trong đó. Để chỉ định bảng nào được dùng, gán property DataMember của DataGrid bằng tên của bảng. Trong ví dụ, chúng ta gán tên là Customers, là tên trùng với tên trong tham số thử hai trong phương thức Fill() của SqlDataAdapter. Đây là lý do tôi thích đặt tên bảng trong phương thức Fill(), và nó giúp đoạn mã dễ đọc hơn.

## Cập nhật thay đổi

Sau khi thay đổi được thực hiện trên dữ liệu, bạn sẽ cần ghi lại vào database. Dòng mã sau cho thấy cách dùng phương thức Update của SqlDataAdapter để cập nhật các thay đổi vào database.

*daCustomers.Update(dsCustomers, “Customers”);*

Phương thức Update() trên được gọi trên thể hiện của SqlDataAdapter có tham số đầu tiên là chính đối tượng gọi phương thức. Tham số thử hai của phương thức Update() chỉ ra bảng nào trong DataSet sẽ được cập nhật. Bảng chứa một danh sách các dòng dữ liệu đã bị thay đổi và các property Insert, Update, Delete của SqlDataAdapter chứa các lệnh SQL dùng để thực hiện thay đối database.

## Kết hợp tất cả lại

Tới bây giờ, bạn đã thấy các phần cần thiết để quản lý disconnected data. Cái bạn cần là xem tất cả công việc trong một ứng dụng. Listing 1 cho thấy cách đoạn mã từ các phần trước được dùng trong một chương trình hoàn chỉnh:

##### Listing 1: Implementing a Disconnected Data Management Strategy

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77 | using System;  using System.Data;  using System.Data.SqlClient;  using System.Drawing;  using System.Windows.Forms;    class DisconnectedDataform : Form  {      private SqlConnection  conn;      private SqlDataAdapter daCustomers;        private DataSet  dsCustomers;      private DataGrid dgCustomers;        private const string tableName = "Customers";        // initialize form with DataGrid and Button      public DisconnectedDataform()      {          // fill dataset          Initdata();            // set up datagrid          dgCustomers = new DataGrid();          dgCustomers.Location = new Point(5, 5);          dgCustomers.Size = new Size(              this.ClientRectangle.Size.Width - 10,              this.ClientRectangle.Height - 50);          dgCustomers.DataSource = dsCustomers;          dgCustomers.DataMember = tableName;            // create update button          Button btnUpdate = new Button();          btnUpdate.Text = "Update";          btnUpdate.Location = new Point(              this.ClientRectangle.Width/2 - btnUpdate.Width/2,              this.ClientRectangle.Height - (btnUpdate.Height + 10));          btnUpdate.Click += new EventHandler(btnUpdateClicked);            // make sure controls appear on form          Controls.AddRange(new Control[] { dgCustomers, btnUpdate });      }        // set up ADO.NET objects      public void Initdata()      {          // instantiate the connection          conn = new SqlConnection(              "Server=(local);DataBase=Northwind;Integrated Security=SSPI");            // 1. instantiate a new DataSet          dsCustomers = new DataSet();            // 2. init SqlDataAdapter with select command and connection          daCustomers = new SqlDataAdapter(              "select CustomerID, CompanyName from Customers", conn);            // 3. fill in insert, update, and delete commands          SqlCommandBuilder cmdBldr = new SqlCommandBuilder(daCustomers);            // 4. fill the dataset          daCustomers.Fill(dsCustomers, tableName);      }        // Update button was clicked      public void btnUpdateClicked(object sender, EventArgs e)      {          // write changes back to DataBase          daCustomers.Update(dsCustomers, tableName);      }        // start the Windows form      static void Main()      {          Application.Run(new DisconnectedDataForm());      }  } |

Phương thức Initdata() trong Listing 1 chứa các phương thức cần thiết để thiết lập SqlDataAdapter và DataSet. Lưu ý rằng các đối tượng dữ liệu khác nhau được định nghĩa tại cấp lớp vì thế chúng có thể được dùng trong nhiều phương thức. Property DataSource của DataGrid được gán trong constructor. Khi người dùng click nút Update, phương thức Update() trong sự kiện btnUpdateClickedsẽ được gọi, cập nhật thay đổi vào lại database.

## Tổng kết

DataSet chứa nhiều bảng và có thể lưu trong bộ nhớ để tái sử dụng. SqlDataAdapter cho phép bạn đổ dữ liệu vào một DataSet và cập nhật thay đổi vào database. Bạn không cần lo lắng về việc mở và đóng SqlConnection bởi vì SqlDataAdapter tự động làm việc đó. Một SqlCommandBuilder tạo ra các câu lệnh insert, update, delete dựa trên câu select của SqlDataAdapter. Dùng phương thức Fill() của SqlDataAdapter để đổ dữ liệu vào DataSet. Gọi phương thức Update() của SqlDataAdapter để cập nhật thay đổi vào lại database.