{C#} STATION

Lesson 02: The SqlConnection Object

This lesson describes the SqlConnection object and how to connect to a database. Here are the objectives of this lesson:

- Know what connection objects are used for.
- Learn how to instantiate a SqlConnection object.
- Understand how the SqlConnection object is used in applications.
- Comprehend the importance of effective connection lifetime management.

Introduction

The first thing you will need to do when interacting with a database is to create a connection. The connection tells the rest of the ADO.NET code which database it is talking to. It manages all of the low-level logic associated with the specific database protocols. This makes it easy for you because the most work you will have to do in code instantiates the connection object, open the connection, and then close the connection when you are done. Because of the way that other classes in ADO.NET are built, sometimes you don't even have to do that much work.

Although working with connections is very easy in ADO.NET, you need to understand connections in order to make the right decisions when coding your data access routines. Understand that a connection is a valuable resource. Sure, if you have a stand-alone client application that works on a single database one machine, you probably don't care about this. However, think about an enterprise application where hundreds of users throughout a company are accessing the same database. Each connection represents overhead and there can only be a finite amount of them. To look at a more extreme case, consider a Web site that is being hit with hundreds of thousands of hits a day Applications that grab connections and don't let them go can have serious negative impacts on performance and scalability.

Creating a SqlConnection Object

A SqlConnection is an object, just like any other C# object. Most of the time, you just declare and instantiate the SqlConnection all at the same time, as shown below:

```
SqlConnection conn = new SqlConnection(
    "Data Source=(local);Initial Catalog=Northwind;Integrated
Security=SSPI");
```

The SqlConnection object instantiated above uses a constructor with a single argument of type string This argument is called a connection string. Table 1 describes common parts of a connection string.

Table 1. ADO.NET Connection Strings contain certain key/value pairs for specifying how to make a database connection. They include the location, name of the database, and security credentials.

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 to make the connection with user's Windows login
User ID	Name of user configured in SQL Server.
Password	Password matching SQL Server User ID.

Integrated Security is secure when you are on a single machine doing development. However, you will often want to specify security based on a SQL Server User ID with permissions set specifically for the application you are using. The following shows a connection string, using the User ID and Password parameters:

```
SqlConnection conn = new SqlConnection(
    "Data Source=DatabaseServer;Initial Catalog=Northwind;User
ID=YourUserID;Password=YourPassword");
```

Notice how the Data Source is set to DatabaseServer to indicate that you can identify a database located on a different machine, over a LAN, or over the Internet. Additionally, User ID and Password replace the Integrated Security parameter.

Using a SqlConnection

The purpose of creating a SqlConnection object is so you can enable other ADO.NET code to work with a database. Other ADO.NET objects, such as a SqlCommand and a SqlDataAdapter take a connection object as a parameter. The sequence of operations occurring in the lifetime of a SqlConnection are as follows:

1. Instantiate the SqlConnection.
2. Open the connection.
3. Pass the connection to other ADO.NET objects.
4. Perform database operations with the other ADO.NET objects.
5. Close the connection.

We've already seen how to instantiate a SqlConnection. The rest of the steps, opening, passing, using, and closing are shown in Listing 1.

Listing 1. Using a SqlConnection

```csharp
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);
```

```csharp
            //
            // 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();
            }
        }
    }
}
```

As shown in Listing 1, you open a connection by calling the *Open()* method of the *SqlConnection* instance, *conn*. Any operations on a connection that was not yet opened will generate an exception. So, you must open the connection before using it.

Before using a *SqlCommand*, you must let the ADO.NET code know which connection it needs. In Listing 1, we set the second parameter to the *SqlCommand* object with the *SqlConnection* object, *conn*. Any operations performed with the *SqlCommand* will use that connection.

The code that uses the connection is a *SqlCommand* object, which performs a query on the Customers table. The result set is returned as a*SqlDataReader* and the

while loop reads the first column from each row of the result set, which is the CustomerID column. We'll discuss the *SqlCommand* and *SqlDataReader* objects in later lessons. For right now, it is important for you to understand that these objects are using the *SqlConnection* object so they know what database to interact with.

When you are done using the connection object, you must close it. Failure to do so could have serious consequences in the performance and scalability of your application. There are a couple points to be made about how we closed the connection in Listing 1: the *Close()*method is called in a *finally* block and we ensure that the connection is not null before closing it.

Notice that we wrapped the ADO.NET code in a *try/finally* block. As described in [Lesson 15: Introduction to Exception Handling](#) of the C# Tutorial, *finally* blocks help guarantee that a certain piece of code will be executed, regardless of whether or not an exception is generated. Since connections are scarce system resources, you will want to make sure they are closed in *finally* blocks.

Another precaution you should take when closing connections is to make sure the connection object is not *null*. If something goes wrong when instantiating the connection, it will be *null* and you want to make sure you don't try to close an invalid connection, which would generate an exception.

This example showed how to use a SqlConnection object with a SqlDataReader, which required explicitly closing the connection. However, when using a disconnected data model, you don't have to open and close the connection yourself. We'll see how this works in a future lesson when we look at the SqlDataAdapter object.

Summary

SqlConnection objects let other ADO.NET codes know what database to connect to and how to make the connection. They are instantiated by passing a connection string with a set of key/value pairs that define the connection. The steps you use to manage the lifetime of a connection are created, open, pass, use, and close. Be sure to close your connection properly when you are done with it to ensure you don't have a connection resource leak.

I hope you enjoyed this lesson and invite you to view the next one in this series, Lesson 03: The SqlCommand Object.

Follow Joe Mayo on Twitter.

Feedback

```
You're Allowed To Do Anything You Want In This
Game!
```

C# Generics: Introduction to Generic Collections in Lesson 20 - C# Station

C# Station

How To: Fetching Web Pages with HTTP - C# Station

C# Station

admin / June 7, 2016 / Lessons / C# database applications, C# Libraries, c# station, software technology, SqlConnection Object

C# Station / Proudly powered by WordPress

Cookies

This website uses cookies to ensure you get the best experience on our website. Learn more.