Objectives

In this chapter, you will:

• Connect to SQL Server from ASP.NET
• Learn how to handle SQL Server errors
• Execute SQL statements with ASP.NET
• Use ASP.NET to work with SQL Server databases and tables
Introduction

- One of ASP.NET’s greatest strengths is its ability to access and manipulate databases
- ASP.NET can access any database that is ODBC compliant
Connecting to SQL Server with ASP.NET

- **Open Database Connectivity (ODBC)**: a standard that allows ODBC-compliant applications to access any data source for which there is an ODBC driver.
- ODBC uses SQL commands to access a database:
  - ODBC then translates the SQL commands into a format that the database understands.
- ASP.NET includes strong support for ODBC.
- ASP.NET also allows you to work directly with SQL Server and Oracle databases:
  - Working directly provides faster access.
Access SQL Server Databases with ASP.NET

• **ActiveX Data Objects (ADO):** a Microsoft database connectivity technology that allows ASP and other Web development tools to access ODBC- and OLE-compliant databases

• **OLE DB:** a data source connectivity standard promoted by Microsoft
  - Supports both relational and nonrelational data sources

• **ADO.NET:** most recent version of ADO that allows access to OLE DB-compliant data sources and XML
Access SQL Server Databases with ASP.NET (cont’d.)

• **Microsoft Data Access Components (MDAC):** components that make up Microsoft’s Universal Data Access technology
  – Include ADO and OLE DB
• MDAC is installed with many Microsoft products, including Internet Explorer, Internet Information Services, Visual Studio, and the .NET Framework SDK
Understanding the System.Data.SqlClient Namespace

- Use classes in the System.Data.SqlClient namespace to access and manipulate SQL Server databases
Understanding the `System.Data.SqlClient` Namespace (cont’d.)

<table>
<thead>
<tr>
<th>Object</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DataSet</td>
<td>Represents data retrieved from a data source</td>
</tr>
<tr>
<td>SqlCommand</td>
<td>Executes a command, such as a SQL command, against a SQL Server database</td>
</tr>
<tr>
<td>SqlConnection</td>
<td>Provides access to a SQL Server database</td>
</tr>
<tr>
<td>SqlDataAdapter</td>
<td>Controls the interaction of a DataSet object with a SQL Server database</td>
</tr>
<tr>
<td>SqlDataReader</td>
<td>Returns read-only, forward-only data from a SQL Server database</td>
</tr>
<tr>
<td>SqlException</td>
<td>Represents the exception that is thrown when an error or warning is returned from SQL Server</td>
</tr>
</tbody>
</table>

**Table 8-1 Core ADO.NET objects**
Connecting to an SQL Server Database

- **SqlConnection** class: used to connect to an SQL Server database
  - Create an object from this class, passing in a connection string
- Connection string must include the **Data Source** parameter with the name of the SQL Server instance you wish to use
Connecting to an SQL Server Database (cont’d.)

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BeginTransaction()</td>
<td>Begins a transaction</td>
</tr>
<tr>
<td>ChangeDatabase()</td>
<td>Changes the currently opened database</td>
</tr>
<tr>
<td>Close()</td>
<td>Closes a data source connection</td>
</tr>
<tr>
<td>CreateCommand()</td>
<td>Creates and returns a Command object associated with the SqlConnection object</td>
</tr>
<tr>
<td>GetSchema()</td>
<td>Returns schema information from the data source</td>
</tr>
<tr>
<td>Open()</td>
<td>Opens a data source connection</td>
</tr>
<tr>
<td>ClearPool()</td>
<td>Empties the SqlConnection object pool for the specified connection</td>
</tr>
<tr>
<td>ClearAllPool()</td>
<td>Empties all SqlConnection object pools</td>
</tr>
</tbody>
</table>

**Table 8-2: SqlConnection class methods**
Connecting to an SQL Server Database (cont’d.)

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ConnectionString</td>
<td>The string used to open a SQL Server database</td>
</tr>
<tr>
<td>ConnectionTimeout</td>
<td>The time to wait before abandoning a SQL Server database connection attempt</td>
</tr>
<tr>
<td>Database</td>
<td>The name of the current SQL Server database to use after a connection has been established</td>
</tr>
<tr>
<td>DataSource</td>
<td>The name of the SQL Server instance</td>
</tr>
<tr>
<td>ServerVersion</td>
<td>The SQL Server version to which the database is connected</td>
</tr>
<tr>
<td>State</td>
<td>A string indicating the current status of the SQL Server database connection</td>
</tr>
</tbody>
</table>

**Table 8-3: SQLConnection class properties**
Opening and Closing a Data Source

• After creating a `SqlConnection` object, use the `Open()` method to open the specified SQL Server database instance

• Use the `Close()` method to disconnect the database connection
  – Database connections do not automatically close when an ASP.NET program ends
Selecting a Database

- Use the **Database** parameter in the connection string to select the database to be used
- Can also select or change a database with the `ChangeDatabase()` method of the `SqlConnection` class
Handling SQL Server Errors

• Must handle situations that occur when you cannot connect to a database server
• Connection may fail because:
  – The database server is not running
  – You have insufficient privileges to access the data source
  – You entered an invalid username and password
• Other causes of errors:
  – You are trying to open a nonexistent database
  – You entered an invalid SQL statement
Checking the Database Connection

- Must verify that your program has successfully connected to a database before attempting to use it
- **State** property of the `SqlConnection` class: indicates the current status of the database connection
Checking the Database Connection (cont’d.)

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Broken</td>
<td>The connection is broken</td>
</tr>
<tr>
<td>Closed</td>
<td>The connection is closed</td>
</tr>
<tr>
<td>Connecting</td>
<td>The Connection object is connecting to the data source</td>
</tr>
<tr>
<td>Executing</td>
<td>The connection is executing a command</td>
</tr>
<tr>
<td>Fetching</td>
<td>The connection is retrieving data</td>
</tr>
<tr>
<td>Open</td>
<td>The connection is open</td>
</tr>
</tbody>
</table>

**Table 8-4: SqlConnection class State property values**
Using Exception Handling to Control SQL Server Errors

• Place the `Open()` method within a `try...catch` block to trap connection errors

• `SqlException` class:
  – Part of the `System.Data.SqlClient` namespace
  – Represents the exception that is thrown when SQL Server returns an error or warning
  – `Number` and `Message` properties provide an error code and message for the exception
Using Exception Handling to Control SQL Server Errors (cont’d.)

**Figure 8-1** Error number and message generated by an invalid user ID:

```
Error code 18452: Login failed for user 'don.gosselin'. The user is not associated with a trusted SQL Server connection.
```
Executing SQL Commands through ASP.NET

• `System.Data.SqlClient` namespace contains classes to access and manipulate SQL Server databases:
  - `SqlDataReader` class
  - `SqlCommand` class
Retrieving Records with the SqlDataReader Class

- **SqlCommand** class: used to execute commands against Microsoft SQL Server version 7.0 or later
- Syntax:
  ```csharp
  SqlCommand object = new SqlCommand
    ("command", connection)
  
  - **command** parameter: contains the SQL command to be executed
  - **connection** parameter: represents the SqlConnection object used to connect to the database
Retrieving Records with the SqlDataReader Class (cont’d.)

• **DataReader** object: used to retrieve read-only, forward-only data from a data source

• **Forward-only**: the program can only move forward sequentially through the records in the returned data from the first to the last

• Use a **DataReader** object when you want to read data but not add, delete, or modify records

• **SqlDataReader** class: used to retrieve data from SQL Server
Retrieving Records with the SqlDataReader Class (cont’d.)

• **ExecuteReader()** method of the **SqlCommand** class: creates a **SqlDataReader** object
  – Must assign the **SqlDataReader** object to a variable

• **Read()** method of the **SqlDataReader** class: advances the **SqlDataReader** object to the next record

• **Cursor**: your position within the recordset
  – Initially placed before the first row in the recordset
  – First use of the **Read()** method places the cursor in the first row of the recordset
Retrieving Records with the `SqlDataReader` Class (cont’d.)

![Figure 8-2 Initial cursor position in a `SqlDataReader` object](image)

**Figure 8-2** Initial cursor position in a `SqlDataReader` object
Retrieving Records with the SqlDataReader Class (cont’d.)

• Use the `Read()` method to determine if a next record is available
  – Returns true if there is another row in the recordset

• Field names in a database table are assigned as variables in a `SqlDataReader` object collection
  – Content of each variable changes when the cursor position moves to a new row
Retrieving Records with the SqlDataReader Class (cont’d.)

• Use the `Close()` method of the `SqlDataReader` class to close it when you are finished working with it
  
  - `SqlDataReader` has exclusive access to the connection object
  
  - You cannot access any other commands until the `SqlDataReader` object is closed
**Figure 8-3** Database records returned with the **SqlDataReader** object
Executing SQL Commands with the SqlCommand Object

- **ExecuteNonQuery()** method of the SqlCommand object: executes commands against a database
  - Used for inserting, updating, or deleting rows in a SQL Server database
  - Does not return a recordset of data
Working with Databases and Tables

• ASP.NET can be used to create databases and tables
  – Use the same SQL commands, but execute them with ASP.NET instead of SQL Server Management Studio
• Note that you normally do not use ASP.NET to create databases and tables
Creating and Deleting Databases

• Use the `CREATE DATABASE` statement with the `ExecuteNonQuery()` method to create a new database
  – If database already exists, an error will occur
• Can test if the database exists with the `ChangeDatabase()` method in a `try...catch` block
  – If unsuccessful, can create the database in the `catch` block
• Use the `DROP DATABASE` statement with the `ExecuteNonQuery()` method to delete a database
Creating and Deleting Databases (cont’d.)

**Figure 8-4** Error code and message that prints when you attempt to create a database that already exists.
Creating and Deleting Databases (cont’d.)

• Central Valley Utilities energy efficiency school sample application
  – Uses a database with two tables: students and registration
• New students page registers students with the school
  – Uses RegularExpressionValidator controls to validate the user input
Figure 8-5 Central Valley Utilities energy efficiency school main Web page
Figure 8-6 New Student page
Figure 8-7 New Student page after adding code to create and select the database
Creating and Deleting Tables

• Use the `CREATE TABLE` statement with the `ExecuteNonQuery()` method to create a new table

• Must select the correct database with the `SqlConnection` constructor or with the `ChangeDatabase()` method before executing the `CREATE TABLE` statement

• Can use the `ExecuteReader()` or `ExecuteNonQuery()` methods to determine whether the table already exists
Creating and Deleting Tables (cont’d.)

Figure 8-8 Error code and message that prints when you attempt to create a table that already exists.
Creating and Deleting Tables (cont’d.)

• IDENTITY keyword: used with a primary key to generate a unique ID for each row in a new table
  – First row’s identity value is 1
  – Each subsequent row’s identity value increases by 1
• You can specify a start value and the increment value if desired
• When adding records to a table with an IDENTITY field, do not include a field value for the IDENTITY field
• Use the DROP TABLE statement with the ExecuteNonQuery() function to delete a table
Adding, Deleting, and Updating Records

• Use the INSERT and VALUES keyword with the ExecuteNonQuery() method to add a record
  – Values in the VALUES list must be in the same order in which the fields were defined in the table
  – Specify NULL in any field for which you do not have a value

• Use the BULK INSERT statement and the ExecuteNonQuery() method to add multiple records using data in a local text file
Adding, Deleting, and Updating Records (cont’d.)

• Use the `UPDATE`, `SET`, and `WHERE` keywords with the `ExecuteNonQuery()` method to update records in a table
  – `UPDATE` keyword specifies the table name
  – `SET` keyword assigns values to fields
  – `WHERE` keyword specifies which records to update

• Use the `DELETE` and `WHERE` keywords with the `ExecuteNonQuery()` method to delete records in a table
  – To delete all records in a table, omit the `WHERE` keyword
Figure 8-9  New Student Web page after obtaining a student ID
Summary

• Open Database Connectivity (ODBC) allows ODBC-compliant applications to access any data source for which there is an ODBC driver

• ActiveX Data Objects (ADO) is a technology that allows ASP to access ODBC- and OLE DB-compliant databases

• Use classes in the System.Data.SqlClient namespace to access and manipulate SQL Server databases with ASP.NET

• Use the SqlConnection class to connect to a SQL Server database
Summary (cont’d.)

- Use the `State` property of the `SqlConnection` class to determine the current status of the database connection.
- Use the `SqlException` class to handle errors.
- Use the `SqlCommand` class to execute commands against SQL Server.
- Use the `ExecuteReader()` method with a `DataReader` object to retrieve data from a data source.
- Use the `SqlDataReader` class to retrieve data from a SQL Server database.
Summary (cont’d.)

• Your position with a data reader object is called the cursor.

• Use the `ExecuteNonQuery()` method of the `SqlCommand` class to execute commands against a database.

• Use the `CREATE DATABASE` statement with the `ExecuteNonQuery()` method to create a new database.

• Use the `CREATE TABLE` statement with the `ExecuteNonQuery()` method to create a new table.
Summary (cont’d.)

• Use the **IDENTITY** keyword with a primary key to generate a unique ID for each new row in a table.

• Use the **DROP TABLE** statement with the `ExecuteNonQuery()` method to delete a table.

• Use the **INSERT** and **VALUES** keywords with the `ExecuteNonQuery()` method to add a new record to a table.

• Use the **BULK INSERT** statement with the `ExecuteNonQuery()` method and a local text file to add multiple new records to a table.
Summary (cont’d.)

- Use the `UPDATE`, `SET`, and `WHERE` keywords with the `ExecuteNonQuery()` method to update records in a table.
- Use the `DELETE` and `WHERE` keywords with the `ExecuteNonQuery()` method to delete records in a table.