Unit 12

Working with Data 15.0 SQL & ADO.NET Fundamentals (Text Book Chapter 14)

➢15.0 SQL and ADO.NET Fundamentals

15.0 SQL & ADO.NET Fundamentals

15.1 What is a Database

A database is an integrated collection of data. There are many different strategies for organizing data to facilitate easy access and manipulation of data. Mechanisms for such operations - **storing and organizing data in a manageable and consistent format are provided by a Database Management System (DBMS)**. Users need not worry about how the things are represented internally and how the data is being accessed.

15.2 What is ADO.NET

ADO.NET [Active Data Objects (ADO). The .NET] provides an API for accessing database systems according to the programs. ADO.NET was created for the .NET Framework and it can be said that they are the new generation of Active Data Objects (ADO). The .NET Framework contains several namespaces and classes, which are devoted to database accesses. Microsoft has created separate namespaces that are optimized for working with different data providers (different types of databases). Generally ADO.NET works well with Microsoft's own databases such as Windows SQL Server.

15.3 How to access a Database?

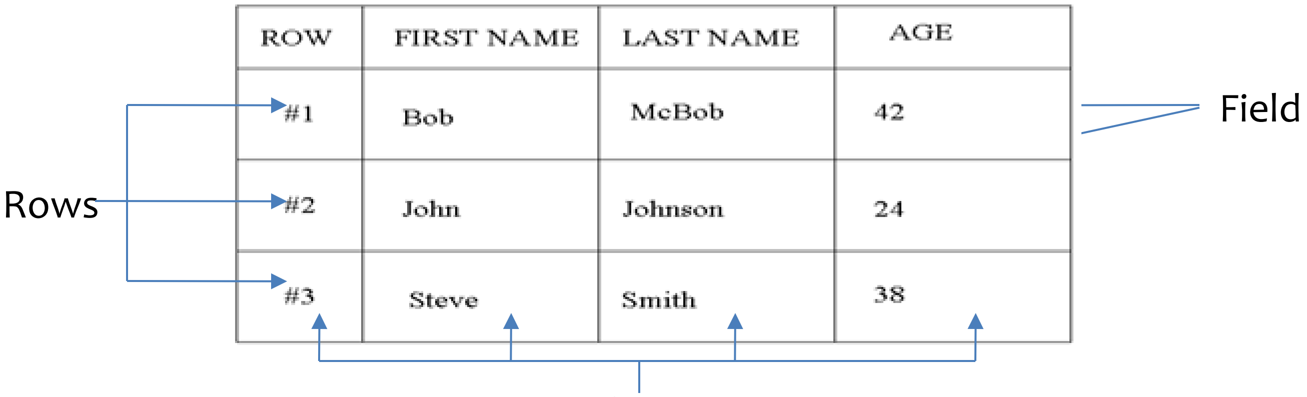
Accessing a database and communicating with the database is carried via a Database Server. For example, Windows SQL Server 2008, MySQL, Oracle are some popular database servers. Behind the database server is the actual data. The Standard language to 'talk' to a database is 'SQL' (or Standard Query Language) With a database server, the client passes SQL requests as messages to the database server. The results of each SQL command are returned over the network. The server uses its own processing power to find the request data instead of passing all the records back to the client and then getting it find its own data. The result is a much more efficient use of distributed processing power. It is also known as SQL engine.

Tips: Make sure you have installed a Database server in your machine. This will help you to try out some examples of this lecture.

15.4 Relational Databases

A relational database is a logical representation of data that allows relationships between the data to be examined without consideration of the physical structure of the data. Relational Databases are composed of tables. In computers, even a normal file of information can be considered as a Database (Rather it is a data source). But they are non-relational. That is you can store data in series of files, where one does not know the content of the other, which could lead to redundant data repetition and anomalies.

At the end, a database refers to a file that is used to store information in a format that is easily retrieved and manipulated. The most common database files are made up of tables, fields and records. In a relational database 'relations' are built between these tables, fields and records



ROW	FIRST NAME	LAST NAME	AGE
#1	Bob	McBob	42
#2	John	Johnson	24
#3	Steve	Smith	38

Field
Rows
Columns

Figure 15.4-1 Example Database Table

15.4.1 Components of a Relational Database

Relational Database consists of the following components…

- A field is a single piece of data.
- A Record is a collection of fields.
- A Table is a collection of records.
- A File is the database in its entirety, i.e. all tables, records and fields. (see Figure 15.4-1)

With the data organized in this way it makes it easier to locate any one piece of information real quick.

➤15.0 SQL and ADO.NET Fundamentals

Figure 15.4-2 shows a set of relations built amongst tables.

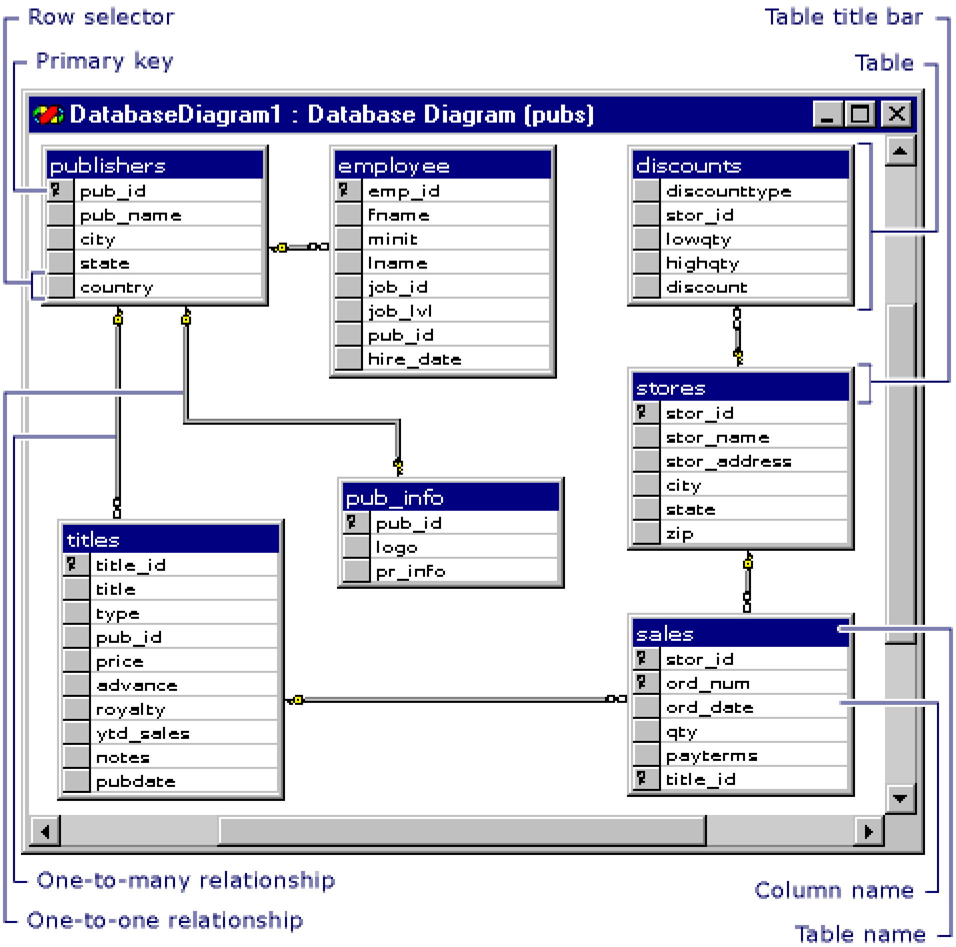


Figure 15.4-2 Relational Tables in a Database

15.5 Structured Query Language (SQL)

15.5.1 Introduction

The operations that you want to carry out on a relational database are expressed in a language that was designed specifically for this purpose, the Structured Query Language – more commonly referred to as SQL. SQL is NOT like a conventional programming language, such as C/C++, C# or Java. SQL is a declarative language – which means the SQL statements tell the SQL server what you want to do but NOT how it should be done. The how is up to the server.

15.5.2 Why SQL?

Structured Query Language (SQL) is accepted internationally as the official standard for relational database access. A major reason for the acceptance of SQL as the relational query language was the move towards client/server architectures that began in the late 1980's.

Some of the 'beauties' of the language are…..

- Easily Readable (like English)
- Syntax is easy to learn
- Construct concepts are very easy to grasp
- SQL always issue commands.

15.5.3 SQL Data Types

The types for data in a relational database are those recognized by the SQL implementation supported by the database engine, and these types will have to be mapped into any language to use the Database. The following chart (Figure 15.5-1) shows the standard SQL data types.

SQL Data Type	Description
CHAR	Fixed length string of characters
VARCHAR	Variable Length String of character
BOOLEAN	Logical Value, true or false
SMALLINT	Small integer value , from −127 to +128
INTEGER	Larger integer value, from −32767 to +32767
NUMERIC	A numeric value with a given precision, which is the number of decimal digits in the number, and a given scale, which is the number of digits after the decimal point. For instance, the value 234567.89 has a precision of 8 and a scale of 2.
FLOAT	Floating point value.
CURRENCY	Stores monetary values
DOUBLE	Higher precision floating point value
DATE	Date

Figure 15.5-1 SQL Data Types

Note : 1. **VARCHAR** is not available in all databases.

2. **NULL** represents the absence of a value of any SQL type of data, but it is not the same as the **null** we have been using in c# or Java.

15.5.4 SQL Statements

Most SQL statements, and certainly the ones we will be using, fall neatly into two categories.

- **Data Definition Language (DDL)** – Statements that are used to describe the tables and the data they contain.
- **Data Manipulation Language (DML)** – Statements that are used to operate on data in a database. DML can be further divided into two groups:

  1. **SELECT** statements – statements that return a set of results
  2. Everything else – statements that don't return a set of results.

Example: Assume we want to create a **"Book Database".** A **table** named **'books'** need to be crated. To make long stories short, lets assume we have only three columns in the books table names 'isbn', 'title' and 'pubcode' as shown in figure 15.5-1

Column Heading	Description
isbn	ISBN is a globally unique identifier for a book
title	Title of the book
pubcode	Code identifying the publisher of the book

Figure 15.5-1 Specification for ‘isbn’, ‘title’ and ‘pubcode’ columns for the ‘book’ table

Column Heading	Data Type
isbn	VARCHAR
title	VARCHAR
pubcode	CHAR(8)

Figure 15.5-2 Data types of ‘book’ table columns

In order to create the table in this example, we will use DDL, which defines syntax for commands such as CREATE TABLE an ALTER TABLE. We will use DDL statements to define the structure of the database. To carry out operations on the database, adding rows to a table or searching the date for instance, we would use DML statements. SQL ‘Create’ is a typical DDL statement.

15.5.4.1 The CREATE statement.

The SQL ‘CREATE’ statement is used below to create the database table. Here we assume that the database is created and the following statement will create the table ‘books’ in the database.

```
CREATE TABLE books (isbn VARCHAR  NOT NULL PRIMARY KEY,
                    title  VARCHAR NOT NULL, pubcode CHAR(8));
```

After executing the above statement, we can think of a table created as shown in figure 15.5-1, but empty (no data yet).

isbn	title	pubcode

Figure 15.5-1 – Table ‘books’ created. But no data yet.

➤15.0 SQL and ADO.NET Fundamentals

Note : The **NOT NULL PRIMARY KEY** for the **isbn** column tells the database two things.

1. No row of the table is allowed to contain a **NULL** value in this column. Every row in this column must always contain a valid value.
2. Because this column is the primary key field, the database should create a unique index in the **isbn** column. This ensures that there will be no more than one row with any given **isbn ID**.

This greatly assists the database when searching through and ordering records. Think how difficult would be to search for an entry in an encyclopedia without an index of unique values in one of the volumes.

Now that we have created a table, we need to put data into the table. The **SQL INSERT** statement does exactly that.

15.5.4.2 The INSERT statement.

There are three basic parts in the INSERT statement.

1. Define the target table for inserting data
2. Define the columns that will have values assigned
3. Define the values for those columns.

An insert statement begins with the keywords INSERT INTO, followed by the name of the target table , list of names of the columns between parenthesis and then the keyword VALUES and then the actual values between parenthesis.

```
INSERT INTO books (isbn, title, pubcode)
   VALUES ('12-34-543-1534Q', 'C# Programming', 'PRENHALL');
```

So we just added one row of information to the book table.

Variations of the INSERT INTO statements are available. For example we can make the above INSERT INTO a shorter one by the following statement. For larger tables it is good to have the columns names too because it acts as a "Navigator" for the builder.

```
INSERT INTO books
 VALUES (‘31-99-564-6554L’, ‘Beginning Linux Programming’, ‘WROX’);
```

will have the same effect as

```
INSERT INTO books (isbn, title, pubcode)
 VALUES (‘31-99-564-6554L’, ‘Beginning Linux Programming’, ‘WROX’);
```

After several more insertions, we can think of a table like the one shown in Figure 16.5. Now that we have entered data into the books table, we need to retrieve data from the Database. We use the ‘SELECT” statement.

isbn	title	pubcode
12-34-543-1534Q	C# Programming	PRENHALL
31-99-564-6554L	Beginning Linux Programming	WROX
78-23-987-561-J	Java Script Professional	PRENHALL
98-32-198-377-M	Managing Risk	PEARSON

Figure 15.5-1 – A Snap of the ‘books’ table after data insertion

15.5.4.3 The SELECT statement

You use the SELECT statement to retrieve information from a database. There are four parts to an SQL SELECT statement:

1. Defining what you want to retrieve
2. Defining where you want to get it from
3. Defining the conditions for retrieval – joining tables, and record filtering
4. Defining the order in which you want to see the data.

For all these we use the **SELECT** statement. Unsurprisingly, the first keyword in the SELECT statement is SELECT. **Then you have several variations to retrieve data the way you want.** SELECT statement tells the database that we need to get some data back in **the form of a result set or a record set.** A result set is just a table of data with fixed numbers of columns and rows – this is some subset of data from a database table generated as a result of the SELECT statement.

➢15.0 SQL and ADO.NET Fundamentals

The **FROM** clause is almost always the companion of the **SELECT** statement. For example

SELECT isbn, title FROM books

means to select columns **isbn** and **title** from the **books table**. With this statement **we will receive a result set like this** (Figure 15.5-2)

isbn	title
12-34-543-1534Q	C# Programming
31-99-564-6554L	Beginning Linux Programming
78-23-987-561-J	Java Script Professional
98-32-198-377-M	Managing Risk

Figure 15.5-2– Using ‘SELECT’ to retrieve ‘isbn’ & ‘title’ data

When a Result Set is retrieved from the database, each Result Set column has a label that, by default, is derived from the column names in the **SELECT** statement.

If you want to select all columns from a table, you can use the *. For example

SELECT * FROM books

means select all columns from the books table and put into a result set (Figure 15.5-3).

isbn	title	pubcode
12-34-543-1534Q	C# Programming	PRENHALL
31-99-564-6554L	Beginning Linux Programming	WROX
78-23-987-561-J	Java Script Professional	PRENHALL
98-32-198-377-M	Managing Risk	PEARSON

Figure 15.5-3– Result after using the wild card ‘*’ with ‘SELECT’

15.5.4.4 The WHERE clause

WHERE is optimization clause that you can use with many statements. A common occurrence is that the **WHERE** to use with the **SELECT** clause.

Consider the table given in Figure 15.5-1 and let's assume we apply the following statement.

SELECT * FROM books WHERE pubcode='WROX'

The result shall be the following (Figure 15.5-4)

isbn	title	pubcode
31-99-564-6554L	Beginning Linux Programming	WROX

Figure 15.5-4– Using 'SELECT' with WHERE caluse

There are several others variations the WHERE can be used. But we will not explore all of them in this course. Always you can look into a good SQL documentation set to see other variations of the WHERE clause.

15.5.4.5 The LIKE and NOT LIKE Search Helpers

The **LIKE** and **NOT LIKE** have two search helper symbols. The underscore _ character that looks for one character and the percentage % character that looks for zero or more characters.

SELECT * FROM books WHERE pubcode LIKE 'P%'

Will produce the table shown in Figure 15.5-5

isbn	title	pubcode
12-34-543-1534Q	C# Programming	PRENHALL
78-23-987-561-J	Java Script Professional	PRENHALL
98-32-198-377-M	Managing Risk	PEARSON

Figure 15.5-5– Result after using the LIKE 'P%' with 'SELECT'

15.5.4.6 ORDER BY clause

Another very helpful clause is the ORDER BY clause. ORDER BY – as it says ORDERS the result in ascending or in descending order if the field to order is numeric. By default it orders the result set in ascending order. Text fields are by default ordered in alphabetical order.

```
SELECT * FROM books ORDER BY title
```

Will produce the table shown in Figure 15.5-6

isbn	title	pubcode
31-99-564-6554L	Beginning Linux Programming	WROX
12-34-543-1534Q	C# Programming	PRENHALL
78-23-987-561-J	Java Script Professional	PRENHALL
98-32-198-377-M	Managing Risk	PEARSON

Figure 15.5-6– Using ‘SELECT’ with ORDER BY clause

15.5.4.7 The UPDATE statement

Update statements provide a way of modifying existing data in a table. Update statements are constructed in a similar way to SELECT statements. You first start with the UPDATE keyword, followed by the name of the table you wish to modify and then followed by the SET and WHERE clauses followed by the new values of columns.

For example,

```
UPDATE books SET pubcode = ‘ADDISON’
                                        WHERE isbn = ‘31-99-564-6554L’
```

will update the book record to reflect a change in the publisher code for the book with ISBN number 31-99-564-6554L.

UPDATE statements do not return a result set, as they merely modify data in the database.

15.5.4.8 DELETE Statement

DELETE statements provide a way of deleting a particular rows from tables in a database. Delete statements consist of the **DELETE** keyword, a **FROM** clause and a **WHERE** clause. For example,

```
DELETE FROM books WHERE isbn = ‘31-99-564-6554L’
```

The above statement **deletes the record in the books table with the ISBN value 31-99-564-6554L.** In the case of the books table, there can only be one row with this value, since its primary key is the ISBN.

By now, you should have a reasonably clear idea of:

- The way SQL is constructed
- How to read SQL statements
- How to construct basic SQL statements

Note however that the SQL statements shown in the above examples are the very basic ones. If you want to get some more ‘real stuff’, you might as well look into a Databases book or reference manual.

Now that we have seen the very basics of SQL, we will now try to understand how ADO.NET (next section) can be used to do all what we did in the previous pages via a Web Application.

➤15.0 SQL and ADO.NET Fundamentals

15.6 ActiveX Data Objects.NET (ADO.NET)

15.6.1 Introduction

ADO(ActiveX Data Objects).NET is the data access component for the .NET Framework. ADO.NET leverages the power of XML to provide disconnected access to data. **ADO.NET is made of a set of classes that are used for connecting to a database, providing access to relational data, XML, and application data, and retrieving results.** ADO.NET data providers contain classes that represent the provider's **Connection, Command, DataAdapter** and **DataReader** objects (among others).

ADO.NET makes it possible a client to establish a connection with a data source, send queries and update statements to the data source, and process the results.

15.6.2 ADO.NET Components

ADO.NET has several key components : **Application Component, DataSet, DataReader** and **ADO.NET Data Provider**. The following Figure 15.6-1 shows a scenario how the application and the ADO.NET data provider are linked.

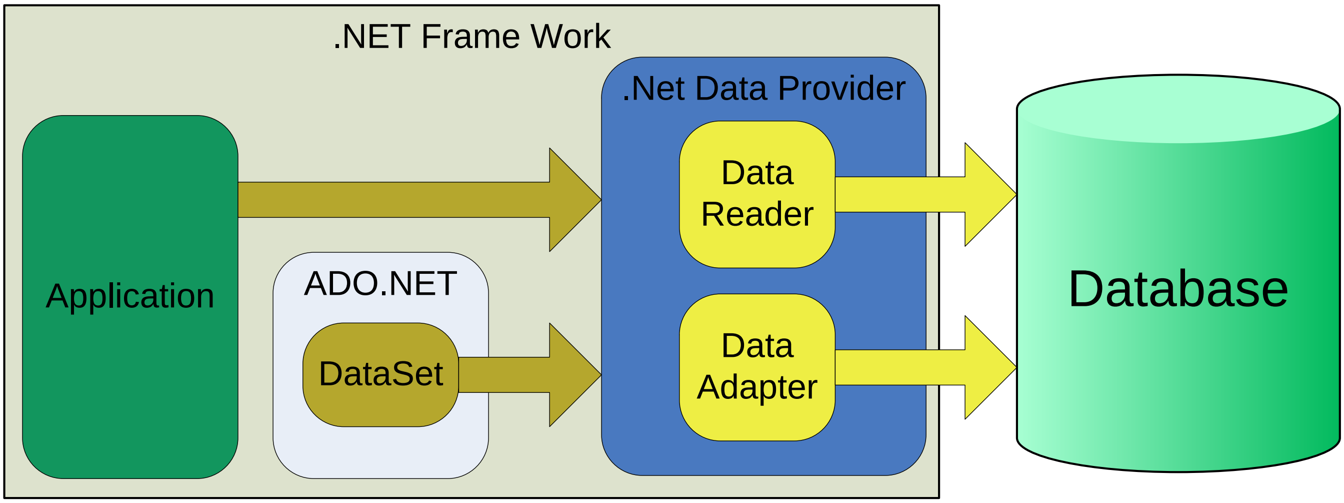


Figure 15.6-1– ADO.Net Components

- **Application component :** This component Processes and calls ADO.NET functions to submit SQL statements and retrieve results.

➢15.0 SQL and ADO.NET Fundamentals

- **DataSet Component :** As in-memory cache of data which functions like a disconnected XML data cache. The overall functions of the DataSet closely recall those of an in-memory database. The DataSet is designed to run in the application space wherever the logic requires local data. This helps increase scalability of systems by reducing load on the major database backends and enabling local processing of data across whatever tier the application requires. For flexibility, the DataSet provides XML and relational interfaces of the data to the developer.

- **DataReader Component**

which provides a direct, read-only SQL interface to the backend. The DataReader is a component of the data provider.

- **ADO.NET Data Provider Component**

This component connects an ADO.NET application to the backend data store. The data provider comprises the Connection, Command, DataReader and DataAdapter objects. The data provider supplies connection information through the Connection object.

Data Providers : Some of the Data Providers used in the industry today are listed below:

SQL Server Provider	: Provides optimized access to a SQL Server Database.
OLE DB Provider	: Provides access to any data source that has an OLE DB driver
Oracle Provider	: Provides optimized access to an Oracle Database
ODBC Provider	: Provides access to any data source that has an ODBC (Open Database Connectivity) driver.

➢15.0 SQL and ADO.NET Fundamentals

15.6.3 AOD.NET Namespaces

ADO.NET managed code components are scattered into several namespaces in the .NET Class library. Together the classes in these namespaces provides the functionality of the ADO.NET. The table 15.6-2 shows the namespaces.

Namespace	Purpose/Description
System.Data	The System.Data namespace provides access to classes that represent the ADO.NET architecture. ADO.NET lets you build components that efficiently manage data from multiple data sources.
System.Data.Common	The System.Data.Common namespace contains classes shared by the .NET Framework data providers.
System.Data.OleDb	The System.Data.OleDb namespace is the .NET Framework Data Provider for OLE DB. The .NET Framework Data Provider for OLE DB describes a collection of classes used to access an OLE DB data source in the managed space. Using the OleDbDataAdapter, you can fill a memory- resident DataSet, which you can use to query and update the data source.
System.Data.SqlClient	The System.Data.SqlClient namespace is the.NET Framework Data Provider for SQL Server. The.NET Framework Data Provider for SQL Server describes a collection of classes used to access a SQL Server database in the managed space. Using the SqlDataAdapter, you can fill a memory-resident DataSet that you can use to query and update the database.
System.Data.SqlTypes	The System.Data.SqlTypes namespace provides classes for native data types in SQL Server. These classes provide a safer, faster alternative to the data types provided by the .NET Framework common language runtime (CLR). Using the classes in this namespace helps prevent type conversion errors caused by loss of precision. Because other data types are converted to and from SqlTypes behind the scenes, explicitly creating and using objects within this namespace also yields faster code.

Figure 15.6-2– ADO.NET Namespaces and their Descriptions

➢15.0 SQL and ADO.NET Fundamentals

15.6.4 AOD.NET Data Provider Classes

A data provider in the .NET Framework serves as a bridge between an application and a data source. A data provider is used to retrieve data from a data source and to reconcile changes to that data back to the data source. The following table (Figure 15.6-3) lists the .NET Framework data providers that are included in the .NET Framework.

.NET Framework data provider	Description
.NET Framework Data Provider for SQL Server	For Microsoft® SQL Server™ version 7.0 or later.
.NET Framework Data Provider for OLE DB	For data sources exposed using OLE DB.
.NET Framework Data Provider for ODBC	For data sources exposed using ODBC. Note The .NET Framework Data Provider for ODBC is not included in the .NET Framework version 1.0. If you require the .NET Framework Data Provider for ODBC and are using the .NET Framework version 1.0, you can download the .NET Framework Data Provider for ODBC at http://msdn.microsoft.com/downloads. The namespace for the downloaded .NET Framework Data Provider for ODBC is Microsoft.Data.Odbc.
.NET Framework Data Provider for Oracle	For Oracle data sources. The .NET Framework Data Provider for Oracle supports Oracle client software version 8.1.7 and later. Note The .NET Framework Data Provider for Oracle is not included in the .NET Framework version 1.0. If you require the .NET Framework Data Provider for Oracle and are using the .NET Framework version 1.0, you can download the .NET Framework Data Provider for Oracle at http://msdn.microsoft.com/downloads.

Figure 15.6-3– ADO.NET Data Providers

The **Connection, Command, DataReader, and DataAdapter** objects represent the core elements of the .NET Framework's data provider model. The following table (Figure 15.6-4) describes these objects.

Object	Description
Connection	Establishes a connection to a specific data source.
Command	Executes a command against a data source.
DataReader	Reads a forward-only, read-only stream of data from a data source.
DataAdapter	Populates a DataSet and resolves updates with the data source.

Figure 15.6-4– ADO.NET Data Provider Classes

➢15.0 SQL and ADO.NET Fundamentals

15.6.5 ADO.NET Data Provider Classes per Provider

NET framework has defined a set of classes for these services (Connection, Command, DataReader, and DataAdapter) for each provider. The table in Figure 15.6-7 lists these classes.

	SQL Server Data Provider	OLE DB Data Provider	Oracle Data Provider	ODBC Data Provider
Connection	SqlConnection	OleDbConnection	OracleConnection	OleDbConnection
Command	SqlCommand	OleDbCommand	OracleCommand	OleDbCommand
DataReader	SqlDataReader	OleDbDataReader	OracleDataReader	OleDbDataReader
DataAdapter	SqlDataAdapter	OleDbDataAdapter	OracleDataAdapter	OleDbDataAdapter

Figure 15.6-5– ADO.NET Data Provider Classes per Provider

15.6.6 Direct Data Access Vs. Caching

A primary choice is whether you want to cache records in a dataset or whether you want to access the database directly and read records using a data reader. For some database operations, such as creating and editing database structures, you cannot use a dataset. For example, if you want to create a new database table from within your application, you cannot do so using a dataset; instead, you would execute a data command. For general data-access scenarios, however, you can often choose between storing records in a disconnected dataset and working directly with the database using data commands.

15.6.6.1Cashing

If you decide to save a dataset between round trips, you must decide where to keep it. This issue is the standard one of state maintenance in Web Forms pages — where do you store information you want to preserve between round trips? For information about saving values, see Web Forms State Management. You have two options:

1. **On the server:** save the dataset **in Session state, Application state**, or **using a cache.**
2. **On the client:** that is, in the page — save the dataset using **view state** or by putting the data into your own **hidden field**. (View state is also implemented using a hidden field.)

15.6.6.2 Direct Data Access Steps

To query information with simple data access, follow these steps.

1. **Create Connection**, Command and DataReader objects.
2. **Use the DataReader** to retrieve information from the database and display it in a control on a web worm.
3. **Close your connection.**
4. **Send the page to the user**. At this point, the information your user sees and the information in the database no longer has not connection and all the ADO.NET objects have been destroyed.

To add or update information, follow these steps

1.Create **new Connection** and Command objects.

2.Execute the command (with appropriate SQL statement).

15.6.6.3 Creating a Connection

The first thing you will need to do when interacting with a data base 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 is instantiate 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 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 seriously negative impacts on performance and scalability.

➢15.0 SQL and ADO.NET Fundamentals

15.6.6.3 Creating a SQL Connection

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 in Figure 15.6.6 describes common parts of a connection string.

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 connection with user's Windows login. [*SSPI stands for the SecuritySupport Provider Interface, which helps a client and server establish and maintain a secure channel, providing confidentiality,integrity, and authentication*]
User ID	Name of user conFigured in SQL Server.
Password	Password matching SQL Server User ID.

Figure 15.6-6– Connection String Components

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");
```

➢15.0 SQL and ADO.NET Fundamentals

UHCL Specific Connection String Components:

UHCL Specific Variables

Data Source =dcm.uhcl.edu (DB Server Name)

Initial Catalog = In your information Slip (Database Name)

User Id = In your information Slip (Your User ID)

Password = your password (Your Password)

Asynchronous Processing=true

[Note : Your personal parameters are were sent via email to you in an information slip early in the semester.]

Example:

```
SqlConnection conn =
new SqlConnection(Data Source=dcm.uhcl.edu;Initial
Catalog=c432013fa01patir;User id= c432013fl;
Password= 9999999;Asynchronous Processing=true);
```

Set the connection setting in web.config file like this using the Information Slip You Received at the start of the semester.

```
<connectionStrings>
    <add name=“made up name for this database"
    connectionString=“User id= c432013fl;
    password= 9999999;
    Data Source =dcm.uhcl.edu;
    Initial Catalog =c432013fl;
    providerName="System.Data.SqlClient“;
    connection timeout=30"/>
</connectionStrings>
```

➤15.0 SQL and ADO.NET Fundamentals

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.

15.6.6.4 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 ofoperations occurring in the lifetime of a SqlConnection are as follows:

-Instantiate the SqlConnection.
-Open the connection.
-Pass the connection to other ADO.NET objects.
-Perform database operations with the other
ADO.NET objects.
-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 Example 15.6.1

➤15.0 SQL and ADO.NET Fundamentals

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

namespace CreatingADBTable
{
 public class CreateATable
 {
  public static void Main(string[] args)
  {
   SqlConnection connectionToServer =
             new SqlConnection(
               "Data Source=(local);Initial Catalog=Northwind;Integrated Security=SSPI");

   string theStatement = "CREATE TABLE Books" +
                      "(isbn CHAR(15), " +
                      "title CHAR(50), " +
                      "pubcode CHAR(1))";

   SqlCommand sqlCommand = new SqlCommand(theStatement, connectionToServer);
   try
   {
    connectionToServer.Open();

    sqlCommand.BeginExecuteNonQuery();

    String insertString = "INSERT INTO Books" +
                      "(isbn, title, pubcode" +
                      "VALUES ('12-34-543-1534Q', 'Java Programming', 'PRENHALL')";
    sqlCommand.CommandText = insertString;
    sqlCommand.BeginExecuteNonQuery();

    insertString = "INSERT INTO Books" +
                "(isbn, title,pubcode" +
                "VALUES ('31-99-564-6554L', 'Beginning Linux Programming', 'WROX')";
    sqlCommand.CommandText = insertString;
    sqlCommand.BeginExecuteNonQuery();

    insertString = "INSERT INTO Books" +
                "(isbn, title,pubcode" +
                "VALUES ('78-23-987-561-J','Java Script Professional', 'PRENHALL')";
    sqlCommand.CommandText = insertString;
    sqlCommand.BeginExecuteNonQuery();

    insertString = "INSERT INTO Books" +
                "(isbn, title,pubcode" +
                "VALUES ('98-32-198-377-M', 'Managing Risk', 'PEARSON')";
    sqlCommand.CommandText = insertString;
    sqlCommand.BeginExecuteNonQuery();

   }
   finally
   {
    connectionToServer.Close();
   }
  }
 }
}
```

Figure 15.6-1– Creating a Table and Inserting Data

15.6.6.5 Using DataReader

A **SqlDataReader** is a type that is good for reading data in the most efficient manner possible. You can *not* use it for writing data. **SqlDataReaders** are often described as fast-forward firehose-like streams of data.

You can read from **SqlDataReader** objects in a forward-only sequential manner. Once you've read some data, you must save it because you will not be able to go back and read it again.

The forward only design of the **SqlDataReader** is what enables it to be fast. It doesn't have overhead associated with traversing the data or writing it back to the data source. Therefore, if your only requirement for a group of data is for reading one time and you want the fastest method possible, the **SqlDataReader** is the best choice. Also, if the amount of data you need to read is larger than what you would prefer to hold in memory beyond a single call, then the streaming behavior of the **SqlDataReader** would be a good choice.

15.6.6.5 Creating a SqlDataReader Object

Getting an instance of a **SqlDataReader** is a little different than the way you instantiate other ADO.NET objects. You must call **ExecuteReader** on a command object, like this:

```
SqlDataReader rdr = cmd.ExecuteReader();
```

The **ExecuteReader** method of the **SqlCommand** object, cmd , returns a **SqlDataReader** instance. Creating a **SqlDataReader** with the new operator doesn't do anything for you. As you learned in previous lessons, the **SqlCommand** object references the connection and the SQL statement necessary for the **SqlDataReader** to obtain data.

15.6.6.6 Reading Data

previous lessons contained code that used a **SqlDataReader**, but the discussion was delayed so we could focus on the specific subject of that particular lesson. This lesson builds from what you've seen and explains how to use the **SqlDataReader**.

➢15.0 SQL and ADO.NET Fundamentals

As explained earlier, the **SqlDataReader** returns data via a sequential stream. To read this data, you must pull data from a table row-by-row. Once a row has been read, the previous row is no longer available. To read that row again, you would have to create a new instance of the **SqlDataReader** and read through the data stream again.

The typical method of reading from the data stream returned by the **SqlDataReader** is to iterate through each row with a while loop. The following code in example 15.6-2a & 15.6-2b shows how to accomplish this.

```
while (rdr.Read())
{
  // get the results of each column
  string isbn = (string)rdr[“isbn"];
  string title = (string)rdr[“title"];
  string pubcode    = (string)rdr[“pubcode"];

  // print out the results
  Console.WriteLine(isbn);
  Console.WriteLine(title);
  Console.WriteLine(pubcode);
  Console.WriteLine();
}
```

Example 15.6-1a– Iterating Data

```
while (rdr.Read())
{
 Book aBook = new Book();
  // get the results of each column
  aBook.setIsbn((string)rdr[“isbn"]);
  aBook.SetTitle((string)rdr[“title"]);
  aBook.SetPublicationCode((string)rdr[“pubcode"]);
}
```

Example15.6-1b– Iterating Data

If you had already created a class named say Book with setXXXX methods such as SetIsbn(). SetTitle and SetPublicationCode(), then you can simple ‘fill’ and object of Book type. See snippet in Figure 15.6.2b.

See a full example next page.

➢15.0 SQL and ADO.NET Fundamentals

In example 15.6-3a a Book class is first created. Then a database table is accessed, read and fills a Book object.

```
public class Book {
  string isbn, title, pubcode;
  public void SetIsbn(string isbn) { this.isbn = isbn; }
  public void SetTitle(string title) { this.title = title; }
  public void SetPublicationCode(string pubcode) { this.pubcode = pubcode;  }
}
```

Example 15.6-2a– A Book class

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

public class ReadingBookTable {
  public static void Main(string[] args) {
    SqlConnection connectionToServer = new SqlConnection( "Data Source=(local);Initial
            Catalog=Northwind;Integrated Security=SSPI");

    string theStatement = "SELECT * from Books";

    SqlCommand sqlCommand = new SqlCommand(theStatement, connectionToServer);
    try {
          connectionToServer.Open();

          SqlDataReader sqlReader = sqlCommand.ExecuteReader();
          while (sqlReader.Read()) {
                                        Book aBook = new Book();

                                        // get the results of each column
                                        aBook.SetIsbn((string)sqlReader["isbn"]);
                                        aBook.SetTitle((string)sqlReader["title"]);
                                        aBook.SetPublicationCode((string)sqlReader["pubcode"]);
                                      }
        }
    finally {   connectionToServer.Close();     }
  }
}
```

Example15.6-2b– Iterate table and 'fill' Book objects

➢15.0 SQL and ADO.NET Fundamentals

15.6.7 Updating a Record in a Table

The **UPDATE** command indicates the records to be modified and the modification to be made. The return value of the **ExecuteNonQuery()** indicates the number of records changes. See the example below.

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

namespace UpdatingATable
{
  public class UpdateATable
  {
    public static void Main(string[] args)
    {
      SqlConnection connectionToServer =
                  new SqlConnection(
                    "Data Source=(local);Initial Catalog=Northwind;Integrated Security=SSPI");

      string theStatement = "UPDATE books SET pubcode =
                                                          'ADDISON' WHERE isbn = '31-99-564-6554L'";

      SqlCommand sqlCommand = new SqlCommand( theStatement, connectionToServer);
      try
      {
       connectionToServer.Open();

       sqlCommand.ExecuteNonQuery();
      }
      finally
      {
       connectionToServer.Close();
      }
    }
  }
}
```

Example 15.6-3 Updating a ‘books’ record

15.6.8 Adding a Record to a Table

The **INSERT INTO** command indicates the records to be added to the table. **ExecuteNonQuery()** indicates the number of records changes. See the example below.

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

namespace InsertIntoATable
{
  public class InsertoToATable
  {
    public static void Main(string[] args)
    {
      SqlConnection connectionToServer =
                  new SqlConnection(
                    "Data Source=(local);Initial Catalog=Northwind;Integrated Security=SSPI");

      string theStatement = "INSERT INTO books (isbn, title, “ +
                          ”pubcode)VALUES ('12-34-543-1534Q', 'C# Programming', 'PRENHALL')";

      SqlCommand sqlCommand = new SqlCommand( theStatement, connectionToServer);
      try
      {
       connectionToServer.Open();

       sqlCommand.ExecuteNonQuery();
      }
      finally
      {
       connectionToServer.Close();
      }
    }
  }
}
```

Example 15.6-4 Adding records to ‘books’ table