Database Programming

ADO.NET Components

ADO.NET stands for "Active Data Objects," where its library is a rich framework that retrieves and updates information in various relational databases, which include Microsoft SQL Server, Microsoft Access, Oracle, and XML. It relies on the .NET Framework's various classes to process requests and transition between a database system and the user (Harwani, 2015, p. 486).

ADO.NET has two (2) components that are commonly used:

- **Data Provider**
  - It is used for connecting in the database, recovering results, updating data, and executing commands.
  - Each database system that ADO.NET supports has a data provider that implements mechanisms for connecting to a database, executing queries, and updating data.
  - .NET Framework currently uses five (5) data providers:
    - **Data Provider for SQL Server** – It gives data access on Microsoft SQL Server.
      - System.Data.SqlClient
    - **Data Provider for OLE DB** – It is used for connecting OLE DB data source.
      - System.Data.OleDb
    - **Data Provider for ODBC**
      - System.Data.Odbc
    - **Data Provider for Oracle**
      - System.Data.OracleClient
    - **EntityClient Provider**
      - System.Data.EntityClient
- **DataSet**
  - It characterizes a local copy of related data tables.
  - It is composed of one or more DataTable objects that are manipulated and updated as per the requirements of the users or client.
  - XML (Extensible Markup Language) data can also work with DataSet.
  - **DataTable** – It manages the data for any entity. It contains zero or more rows of data, and each value shown in the data table is identified by the `DataColumn`. It also consists of primary key, foreign key, constraint, and relation information of different tables.
  - **DataColumn** – It includes a data type declaration based on the kind of data saved.
  - **DataRelation** – It is used to link two (2) `DataTable` classes inside the `DataSet` class.
  - **DataColumnMapping** – It is used to map a table name from the database to a `DataTable` within a `DataSet`.
  - **DataView** – It is used to customize the view of the table or desired view of the rows in a `DataTable`.

Data Provider contains the following objects that define the function to be used in a database management system (DBMS):

- **Command –** It can run SQL commands and perform a stored procedure—accessing and manipulating data. Data management statements and SQL queries are wrapped in the `Command` object.
- **Connection –** From the data source, it allows users to connect and disconnect. Communications with the external data sources happen through the `Connection` object.
- **DataAdapters –** It transfers `DataSets` between the data source and the caller. It also acts as a connection between the data source and `DataSet`, where DataAdapters contain a connection and set of `Command` objects that help in fetching and updating data.
- **DataReaders –** It is a read-only access to data using a server-side cursor. The `DataReader` object is gained by invoking the `ExecuteReader` method of the `Command` object. It requires closing the connection when using the `DataReader` object.

ADO.NET contains classes for connecting or using an SQL command. It requires to add the `System.Data.SqlClient` namespace. See *Table 1* for the database-specific classes contained in ADO.NET.

Database Classes	Description
`SqlCommand`, `OleDbCommand`, and `ODBCCommand`	Used as wrappers for SQL statements or stored procedure calls
`SqlCommandBuilder`, `OleDbCommandBuilder`, and `ODBCCommandBuilder`	Generate SQL commands for updating database tables Automatically generate SQL commands from a `SELECT` statement
`SqlConnection`, `OleDbConnection`, and `ODBCConnection`	Used to connect to the database
`SqlDataAdapter`, `OleDbDataAdapter`, and `ODBCDataAdapter`	Handle interaction between `DataSet` and data source Help in executing different SQL commands to populate a `DataSet` and update the data source
`SqlDataReader`, `OleDbDataReader`, and `ODBCDataReader`	Used as a forward-only, read-only access to data using a cursor
`SqlParameter`, `OleDbParameter`, and `ODBCParameter`	Define a parameter to a stored procedure
`SqlTransaction`, `OleDbTransaction`, and `ODBCTransaction`	Represent transactions to be made in data source

Table 1. Database-specific classes in ADO.NET
Source: Learning Object-Oriented Programming in C# 5.0, 2015, pp. 488–489.

These classes implement a set of standard interfaces defined within the `System.Data` namespace. Refer to the outline of the namespaces included in ADO.NET below according to Harwani (2015, p. 489).

- `System.Data` – It includes all generic data access classes.
- `System.Data.Common` – It includes classes that are shared or overridden by individual data providers.
- `System.Data.EntityClient` – It includes Entity Framework classes.
- `System.Data.Linq.SqlClient` – It includes LINQ to SQL provider classes.
- `System.Data.Odbc` – It includes ODVC provider classes.
- `System.Data.OleDb` – It includes OLE DB provider classes.
- `System.Data.ProviderBase` – It includes new base classes and connection factory classes.
- `System.Data.Sql` – It includes new generic interfaces and classes for SQL Server data access.
- `System.Data.SqlClient` – It includes SQL Server provider classes.
- `System.Data.SqlTypes` – It includes SQL Server data types.

Creating a Database

To connect or create a database using Visual Studio for C#, use any of the two (2) windows below:

- **Server Explorer** – It is a server management console that is used for logging in to servers, opening data connections, and accessing databases.
    - **Data Connections** – It contains all the connections to the server and local databases. The data connection node displays the list of tables, view, stored procedures, and functions. It can also be used to connect to a SQL Server Database.
- **SQL Server Object Explorer** – It is used to design, connect to the available database, browse the schema, and create a query on its object.

To have an SQL Server with an integrated environment for managing any SQL infrastructure, use SQL Server Management Studio (SSMS). It can deploy, monitor, and upgrade the data-tier components used by applications. It can also build queries and scripts.

Creating a Local Database – To create a local database, right-click the project file in the **Solution Explorer**, click **Add New Item**, and choose **Service-Based Database**. The file will be saved as .mdf file (Master Database File or MDF is known as the main database file which contains data and schema). The .ldf file (Log Database Files or LDF) under .mdf file contains logs for each transaction of the database. This allows SQL database to be recoverable in the case of data loss.

Creating a Table – To create a table, go to the **Server Explorer** and look for the database that has been created in the .mdf file format. After that, right-click folder named **Tables** and choose **Add New Table**. A new window will prompt that allows to write a query in T-SQL (Transact-SQL) or use the table editor to create a table. The table name can be renamedin T-SQL. To save the created table for the database, click the **Update** button, which is on the left side of the **Script File** label. A

message will prompt that allows previewing the updates on the database. After that, click the **Update Database** button to add the created table.

Query for creating a database: `CREATE DATABASE db_name;`

Query for dropping a database: `DROP DATABASE db_name;`

Query for creating a database table: `CREATE TABLE table_name ( column1 datatype, column2 datatype, column3 datatype,…);`

Query for dropping a database table: `DROP TABLE table_name;`

Accessing Data from a Database

To access the database, it requires connecting to the database server. Use a `Connection` object to establish a connection to the data source. Refer to the following steps to access data from the database:

1. Create a connection to the specified server.
2. Open a connection using the `SqlConnection` object.
3. Create a command using the `SqlCommand object`.
4. Use and create a DataReader or DataAdapter object.
5. Call the `Close()` method to close the connection and release the resources.

`SqlConnection`

- This class is used to open a connection using the `Open()` method. The connection can be closed by calling the `Close()` method.
- In this class, some of these properties can be used while connecting to the database:
  - **ConnectionString** – a property that is used to read or assign the connection string to be used by the `SqlConnection` class
  - **DataSource** – a read-only property that returns the name of the SQL Server instance
  - **Database** – a read-only property that returns the name of the database
  - **State** – a read-only property that returns the current state of the connection.
- Some attributes are commonly used in a connection string. See *Table 2* for the frequently used attributes in a connection string.

ATTRIBUTE	DESCRIPTION

`Server`	To establish a connection, specify the computer name to which you want to connect. Basically, the computer name where the desired database is installed is specified in the following format:
	`SqlConnection conn = new SqlConnection("Server=computer_name;");`
	If the application is created on the same machine where the database is installed, the computer name can be specified as local:
	`SqlConnection conn = new SqlConnection("Server=(local);");`
	The attribute for specifying the computer name is `Server`, `Data Source`, `Address`, or `Addr`. For example, the preceding statement can also be written as shown below:
	`SqlConnection conn = new SqlConnection("Data Source=(local);");`

`Database / Initial Catalog`	The attribute is used to specify the database to access the specified server. The `Database` keyword can also be substituted for `Initial Catalog`. For example, the following statement establishes a connection with `CSharpSampleDatabase` on the local computer: `SqlConnection conn = new SqlConnection("Server=(local); Initial Catalog=CSharpSampleDatabase;");` This attribute is optional. In other words, it can be ignored when you want to connect to a specific computer but not to any database.
`Trusted_Connection / Integrated Security`	This attribute implements the secure connection with the database. The values `true`, `false`, `yes`, `no`, or `SSPI` can be specified for this attribute. While establishing a trusted connection or the connection that doesn't need to be verified, a value of `true` or `SSPI`

ATTRIBUTE	DESCRIPTION
	(Security Support Provider Interface) can be assigned for this attribute as shown below: `SqlConnection conn = new SqlConnection("Server=(local); Initial Catalog=CSharpSampleDatabase; Integrated Security=no");` After setting the security attribute to `false` or no, provide the login credentials.
`Username`	To specify the username used in a connection after assigning `false` or no to the Integrated Security attribute, use the User ID attribute and assign a valid username to it. Here is an example: `SqlConnection conn =new SqlConnection("Server=(local); Initial Catalog=CSharpSampleDatabase; Integrated Security=no; User ID = David");`
`Password/Pwd`	When establishing a secure connection to a database, in addition to a valid username, a password is needed. `Password` or PWD (not case sensitive) can be used, as shown in the following example: `SqlConnection conn = new SqlConnection("Server=(local); Initial Catalog=CSharpSampleDatabase; Integrated Security=no; User ID=David; PWD=gold2019");`

Table 2. Frequently used attributes in a connection string
Source: Learning Object-Oriented Programming in C# 5.0, 2015, pp. 500–501.

`SqlCommand`

- This class is used to perform desired processing on the database. It allows executing any direct T-SQL or stored procedure on the server.

Methods of the `Command` *object*

- **`ExecuteNonQuery`** – This method executes the command but does not return output.
- **`ExecuteReader`** – This method executes the command and returns a `DataReader` object based on the SQL statement.
- **`SqlDataReader`** – This object is forward-only and read-only cursor that can be iterated through to fetch the rows in it.
- **`ExecuteRow`** – It executes the command and returns the `SqlRecord` object that contains a single returned row.
- **`ExecuteScalar`** – It executes the command and returns the first column of the first row in the result set.

Properties of the `Command` *object*

- **`Connection`** – It gets or sets the `SqlConnection` class that can be used by the `Command` object.
- **`CommandText`** – It gets or sets the stored procedure or the T-SQL statement.
- **`CommandType`** – It indicates the way the `CommandText` property should be interpreted.

Query for selecting all the data of the table: `SELECT * FROM table_name;`
Query for selecting specific columns of the table: `SELECT column1, column2, column3,… FROM table_name;`

Inserting Rows in a Table

To insert rows in a table, use the `INSERT` command, `Command` object, and `Parameter` object.

SqlParameterCollection

- It refers to a collection of parameters that is connected with `SqlCommand`.
- Adding `Parameter` objects to a `SqlParameterCollection` uses two (2) methods:
  - `Add()`
    - It adds the specified `SqlParameter` object to the `SqlParameterCollection` and contains two (2) arguments: string **`paramaterName`** and **`SqlDbType`**.
    - **`SqlDbType`** – It specifies the SQL Server data type of a field and property.
    - Syntax: `public SqlParameter Add(string paramterName, SqlDbtype sqlDbType);`
    - Sample Code: `sqlCommand.Parameters.Add("@FirstName", SqlDbType);`
  - `AddWithValue()`
    - It adds a value to the end of the `SqlParameterCollection`.

- It contains two (2) arguments, a string `parameterName` and an `object`, that includes the value of that parameter.
- Syntax: `public SqlParameter AddWithValue(string parameterName, object value);`
- Sample Code: `sqlCommand.Parameters.AddWithValue("@FirstName", txtBoxFirstName.Text);`

Query for inserting data for all the columns of the table: `INSERT INTO table_name VALUES(value1, value2, value3,… );`

Query for inserting data for specified column names: `INSERT INTO table_name (column1, column2, column3, … ) VALUES(value1, value2, value3, … );`

Updating a Table

To update a table on the database, use the SQL UPDATE command, Command object, and `Parameter` object.

SQL UPDATE

- It is used to modify or update an existing record.
- It uses the `WHERE` clause to determine the number of records that will be updated.
- Query for updating data: `UPDATE table_name SET column1 = value1, column2 = value2, column3 = value3 WHERE condition;`

REFERENCES:

Deitel, P. & Deitel, H. (2015). *Visual C# 2012 how to program* (5th ed.). USA: Pearson Education, Inc.

Doyle, B. (2015). *C# programming: from problem analysis to program design* (5th ed.). Boston, MA: Cengage Learning.

Gaddis, T. (2016). *Starting out with visual C#* (4th ed.). USA: Pearson Education, Inc.

Harwani, B. (2015). *Learning object-oriented programming in C# 5.0.* USA: Cengage Learning PTR.

Miles, R. (2016). *Begin to code with C#.* Redmond Washington: Microsoft Press.