

SCSL

Workbook for ADO.Net

A beginner's guide to effective programming

(In C#.Net)

Why This Module







A computer can be thought of as an information storage and processing machine. It's hard to imagine a computer program that doesn't work with any kind of data. While some applications choose to use server-based database architecture like, Oracle, Microsoft SQL Server and others, certain others might work upon file-based architecture like Microsoft Access.







Databases are designed specially to handle the information as tables, which arrange data as a collection of rows and columns and values within them. However, programming languages have a different method of representing data. In fact, one program could work with more than one data source at a time and it needs some sort of data access library to accomplish this task. There is a mismatch between how most databases handle information and how most programming languages handle information. This is where ADO.Net comes into picture. ADO.Net is designed to provide consistent access to data source.

This book takes a new approach, focusing on the practical tasks like connecting to the database, retrieving data, and working with transactions. It offers the practical understanding, viewpoint, and knowledge developers are looking for. This book explains what is available in ADO.NET by associating it with the need to solve a practical problem and better architect an application, rather than remembering hundreds of classes and properties available in the framework.

Guide to Use this Workbook

Conventions Used:

Convention	Description
 Topic	Indicates the Topic Statement being discussed.
Estimated Time	Gives an idea of estimated time needed to understand the Topic and complete the Practice session.
 Presentation	Gives a brief introduction about the Topic.
 Scenario	Gives a real time situation in which the Topic is used.
 Demonstration/Code Snippet	Gives an implementation of the Topic along with Screenshots and real time code.
<i>Code in Italic</i>	Represents a few lines of code in Italics which is generated by the System related to that particular event.
// OR	Represents a few lines of code (Code Snippet) from the complete program which describes the Topic.
 Context	Explains when this Topic can be used in a particular Application.
 Practice Session	Gives a practice example for the participant to implement the Topic, which gives him a brief idea how to develop an Application using the

	Topic.
 Check list	Lists the brief contents of the Topic.
 Common Errors	Lists the common errors that occur while developing the Application.
 Exceptions	Lists the exceptions which result from the execution of the Application.
 Lessons Learnt	Lists the lessons learnt from the article of the workbook.
 Best Practices	Lists the best ways for the efficient development of the Application.
 Notes	Gives important information related to the Topic in form of a note

Contents

1. Data-Centric Applications and ADO.NET	7-18
1.1 Design of data centric applications	8
1.2 ADO.Net architecture	11
1.3 ADO.Net and XML	15
1.4 Crossword	18
2. Connecting to Data Sources	19-32
2.1 Choosing a .Net data provider	20
2.2 Defining a connection	20
2.3 Managing a connection	20
2.4 Handling connection exceptions	24
2.5 Connection pooling	29
2.6 Crossword	32
3. Performing Connected Database Operations	33-51
3.1 Working in a connected environment	34
3.2 Building command objects	34
3.3 Executing command objects that return single value	34
3.4 Executing commands that return rows	34
3.5 Executing commands that do not return rows	34
3.6 Command Parameters	40
3.7 Executing stored procedures	40
3.8 Using transactions	45
3.9 Transaction vocabulary	45
3.10 Crossword	51
4. Building DataSets	52-78
4.1 Working in a disconnected environment	53
4.2 DataSet Object Model	53
4.3 Building Datasets and Data tables	53
4.4 Binding and saving a Dataset	53

4.5	Defining data relationships	59
4.6	Modifying data in a Data table	63
4.7	Sorting and Filtering	69
4.8	Crossword	78

5. Reading and Writing XML with ADO.NET	79-99
--	--------------

5.1	Creating XSD Schemas	80
5.2	Strongly typed DataSets	87
5.3	Loading schemas and data into Datasets	90
5.4	Writing XML from a Dataset	94
5.5	Crossword	99

6. Building DataSets from Existing Data Sources	100-129
--	----------------





6.1	Configuring a data adapter to retrieve information	101
6.2	Populating a Dataset using a data adapter	107
6.3	Configuring a data adapter to update the underlying data source	107
6.4	Persisting changes to a data source	112
6.5	How to handle conflicts	116
6.6	Typed and UnTyped datasets	122
6.7	Binding to DataGrid control	122
6.8	Crossword	128

****Answers For Crosswords** **129**

****Contributors** **130-131**

1.0 Data-Centric Applications and ADO.NET

Topics

-  1.1 Design of Data Centric Applications
-  1.2 ADO.Net Architecture
-  1.3 ADO.Net and XML
-  1.4 Crossword





Topic: Data Centric Applications

Estimated Time: 20 mins.



Objectives: At the end of the session, the participant will know the :

- Use of storage options.
- Use of different types of data storage and the classes.
- Connected and Disconnected application environment.



Presentation:

ADO.NET supports the following types of data storage:

- Unstructured
- Structured Non-Hierarchical data
- Hierarchical
- Relational Database

Connected Environment:

- Connection to the database will remain open as long as the application is open.
- All changes are done directly to the database and no local (memory) buffer is maintained.

Disconnected Environment:

- Connection is opened to serve the request of an application (write a query) and is closed as soon as the request is completed (once the query is resolved).
- **Local buffer** of persistent data called **DataSet** is managed.

ADO.NET Data Storage Classes

- Data storage in ADO.NET is provided for by real classes.
- DataSet class is the most important class involved in the ADO.NET storage mechanism.
- Primary ADO.NET data storage classes, and their relations, are :

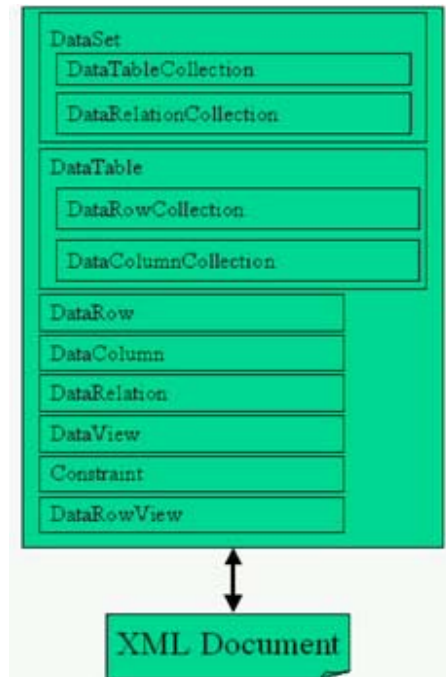


Figure 1.1-1: ADO.Net Data Storage Classes



Scenario:

For Connected Environment - Consider a factory that requires a real-time connection to monitor production output and storage.

For Disconnected Environment - A warehouse manager has a Microsoft Windows CE device running SQL Server CE that he uses to keep track of stock when a batch of products enters or leaves the warehouse.



Context:

- Disconnected environment is used when the amount of data to be queried or updated is less and when there is a need to view persistent data.
- Connected environment allows frequent retrieval or updating large volume of data, from or to the database.



Practice Session:

- Compare Disconnected environment with Connected environment.
- Suggest real-life scenarios wherein Disconnected environment is likely to be employed.
- Suggest real life scenarios wherein connected environment is likely to be employed.



Check List:

- Importance of Connected and Disconnected architecture.
- Importance of Data Storage and types.
- Benefits of Disconnected architecture over Connected architecture.
- Data Storage Classes and its components.



Common Error :

- Not configuring the server before working on the database.



Exception :

- An Exception might occur while committing changes to the database in a disconnected environment if resources like domain controller are not available.



Lessons Learnt :

- ☒ Disconnected architecture improves performance and reduces network traffic.
- ☒ Data storage classes give the ability to store, edit, update, and control the view of data, regardless of how this data was obtained.



Best Practice:

- It is always better to use disconnected architecture for better performance depending on requirements.



Topic: ADO.Net Architecture

Estimated Time: 30 mins.



Objectives: On completion of the activity, the participant will have knowledge on:

- How ADO.NET is divided into namespaces
- Various data access techniques
- Object model involved in ADO.Net.



Presentation:

Namespaces

- Namespaces that allow you to manage and work with data are referred to as **data-related namespaces**.
- **System.Data** is the main data related namespace storing all of ADO.NET architecture's classes.
- Syntax for including System.Data Namespace in VB.net is Imports System.Data
- Syntax for including System.Data Namespace in C#.net is **using System.Data;**
- System.Data namespace contains the following two Data related namespaces:
 - **System.Data.OleDb**
 - **System.Data.SqlClient**

Data Access

- Data Access in ADO.NET relies on two components:
 - DataSet: It is a disconnected, in-memory representation of data.
 - Data Provider: It is responsible for providing and maintaining the connection to the database.
- DataSet is a common method for accessing data and it includes DataTable and DataRelation.
- DataTable object represents a single table in the database having a name, rows, and columns.
- DataRelation is a relationship between tables, such as a primary-key and foreign-key relationship.
- DataProvider is a set of classes like:
 - Connection: provides a connection to the database
 - Command: used to execute a command
 - DataReader: provides a forward-only, read only, connected RecordSet.
 - DataAdapter: populates a disconnected DataSet with data and keeps updating

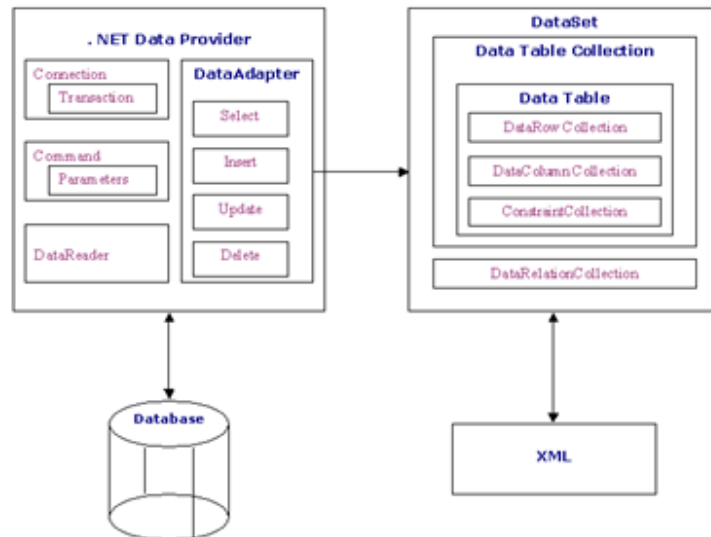


Figure 1.2-1: ADO.Net Architecture

ADO.Net Object Model

- **Connected object model:**

It uses a Connection object for creating a connection, a Command object for executing queries and a DataReader object for establishing a forward-only, read-only connected stream to the database.

- **Disconnected object model:**

It uses a DataSet as a local copy of the relevant portions of the database and a DataAdapter object which serves as a bridge between the DataSet and the Database.

Evolution of ADO to ADO.Net

- ADO was a connected data access.
- An open connection for the lifetime of the application raises concerns about database security and network traffic.
- Open database connections use system resources to a maximum extent making the system performance less effective.
- Hence ADO.Net was introduced to overcome all such difficulties.



Scenario :

Mr. David owns SmartTech Hardware Solutions Company. He wants to view details of his employees which are maintained in MSSQL Server. Build a windows-based application which displays details of all the Employees from the Employee Table. The employee details include EmployeeId , EmployeeName, address, city, state, and phone_no.



Demonstration/Code Snippet :

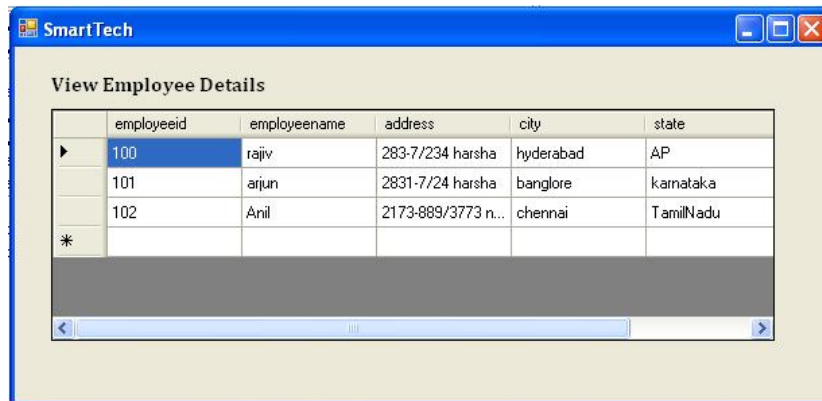


Figure 1.2-2: Output screenshot at the end of execution

Step 1: Open Visual Studio and create a new Windows application named, SmartTech. Insert the following code which includes the corresponding namespaces.

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

Step 2: Insert the following code in the SmartTech_Load event. The following code only declares the variables of connection, string, DataAdapter and DataSet type. Notice that you do not need to specify the namespace for the DataSet and SqlConnection classes:

```
SqlConnection MyCon;
string MyConString;
SqlDataAdapter da;
DataSet ds;
```



Context:

- ADO.Net allows a number of users to access the data from the database while using an application.
- System.Data namespace enables management and access of data from multiple data sources.
- DataSet provides a consistent relational programming model regardless of the source of the

data it contains.

- DataRelation enables DataRow's functionality of retrieving related rows.



Practice Session :

- Compare the various aspects of ADO with ADO.NET.
- Explore the various methods, properties and events listed in the DataAdapter class and DataSet class, from MSDN.



Check List :

- Types of data access techniques.
- Evolution of ADO.Net
- Review through the fundamental objects available in ADO.Net Object Model.



Common Error :

- Incorrect or insufficient namespaces imported into the application.



Exception :

- An Exception might occur if coding is done with respect to a data source which is different from the imported namespace.



Lessons Learnt:

- ☒ In Disconnected environment, data is not always up to date and change conflicts must be resolved.
- ☒ A Disconnected environment improves the scalability and performance of applications.
- ☒ A Connected environment must have a constant network connection.
- ☒ In a Connected environment, a secure environment is easier to maintain and concurrency is easier to control.



Best Practice :

- Usage of appropriate provider object to connect to the corresponding databases



Topic: ADO.Net and XML

Estimated Time: 20 mins.



Objectives: On completion of the activity, the participant will be able to :

- Appreciate the use of XML in ADO.NET
- Understand working of XML Web service with ADO.NET



Presentation:

- ADO.NET is tightly integrated with XML.
- ADO.NET converts relational data into XML format and vice-versa.
- ADO.NET uses XML to transfer data between the server and client.

Importance of XML

- XML is a portable way of representing data in an open and platform independent way??.
- XML data is text-based.

XML support in ADO.NET

- DataSet receives data from a database as XML-based content.
- DataSet updates the new contents to the database using XML.
- Typed DataSets can be created by defining an XML Schema for the XML representation of data in it.

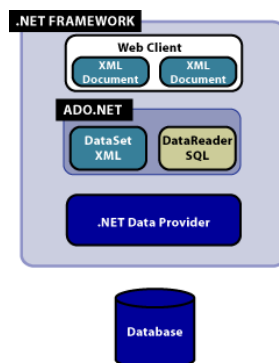


Figure 1.3-1: XML Support in ADO.Net

XML WebService with ADO.NET

- XML Web services can be used internally by a single application or accessed over the Internet and used remotely by multiple applications.
- Client applications invoke an XML Web service, to request data from a database and re-load it.

- Client applications process the returned XML data and bind it to user-interface controls such as a DataGrid.



Scenario:

Satyam Computer Services Ltd. is a leading IT service provider in the world. Newly joined employees are required to submit their profile in E-Support (web application) before their joining date. The details submitted by them are committed to Satyam's server. In such multi-tier applications, data is transported as XML with the database logic constituted by the XML Web service housed on the server.



Context:

- Reading data from a DataSet in XML format is useful in a distributed environment.
- XML format allows re-usability and inter-operability between applications and services.
- An XML Schema is created to perform tasks such as serializing the XML data to a stream or file.



Practice Session:

- Evaluate and explore the utility of XML web service in distributed application development.



Check List:

- XML and its support for ADO.Net.
- Creation and use of XML WebService in Windows Application.



Common Errors:

- Accessing data without establishing connection to the database.
- Reading or writing data from the file without permission.



Exceptions:

- An Exception might occur if the XML web service's proxy class fails to pass the security credentials for Windows authentication.
- SOAP Exception occurs if an error occurs while calling XML Web service method over SOAP.



Lessons Learnt:

- ☒ Treating a database as a large virtual XML document with ADO.NET demands more memory and its performance could be a major problem in the case of complex and enormous databases. But for relatively smaller projects, it works flawlessly and saves a lot of time.
- ☒ XML is used to represent structured data

- ☒ XML WebService is used to transfer data between client and server.

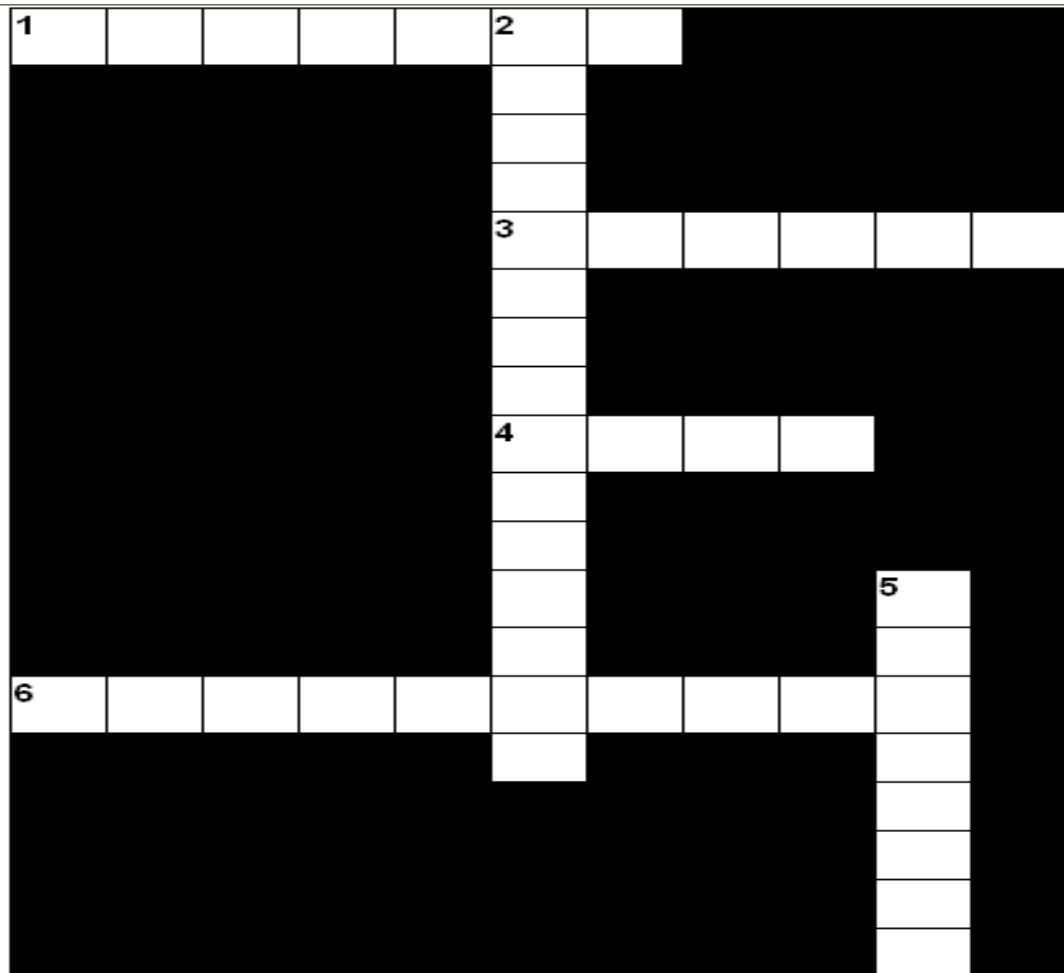


Best Practices:

- Accessing an XML Web service in managed code is better through a proxy class that Visual Studio generates directly from the service description of the XML Web service.
- Proxy class transforms the method call into a request message and the response message back into a method return value.

Crossword: Unit-1

Estimated Time: 10 mins.



Across:

1. It can contain tables and relationships between those tables (7).
3. Properties of DataAdapter(6)
4. Method used to open a connection (4).
6. A _____ describes an enforced property of the database, such as the uniqueness of the values in a primary key column(10)

Down:

2. Commands that have no return values (15).
5. This serves as data binding to Windows Forms and Web Forms(8)

2.0 Connecting to Data Sources

Topics

- + 2.1 Choosing a .Net data provider
- + 2.2 Defining a Connection
- + 2.3 Managing a Connection
- + 2.4 Handling Connection Exceptions
- + 2.5 Connection Pooling
- + 2.6 Crossword





Topic: Connecting to Data Sources

Estimated Time: 30 mins.



Objectives : On completion of the activity, the participant will have knowledge on:

- Choose a .Net data provider
- Connect to SQL SERVER
- Define a Connection.
- Manage a Connection.



Presentation:

Choose a .Net Data Provider

- .NET Framework Data Provider is used for connecting to a database of some particular data source, executing commands, and retrieving results.
- It is designed to be light weight, providing a minimal layer between the data source and code, thereby increasing performance.
- Different types of Data Providers:
 - .NET Framework Data Provider for SQL Server
 - .NET Framework Data Provider for OLE DB
 - .NET Framework Data Provider for ODBC
 - .NET Framework Data Provider for Oracle

Defining a Connection

- Connection is defined using a connection string.
- Connection string specifies information about a data source to connect
- Connection string may include attributes such as the name of the driver, server, database, user name and password
- The types of authentication modes for defining connection are
 - Windows Authentication
 - Mixed Mode Authentication

Managing a Connection

- A connection is managed using a Connection object. The Connection object has the following methods:
 - Open()
 - Close()
- Connection instantiation can be either explicit or implicit
- Dispose () method de-allocates memory for implicit connection



Scenario :

Mr. David owns SmartTech Hardware Solutions Company. He wants to view the details of the employees maintained in MSSQL Server. Build a windows-based application which displays details of all the Employees from the Employee Table. The employee details include EmployeeId , EmployeeName, Address, City, State, Phone_no. Illustrate how the connection can be closed implicitly and explicitly.



Demonstration/Code Snippet :

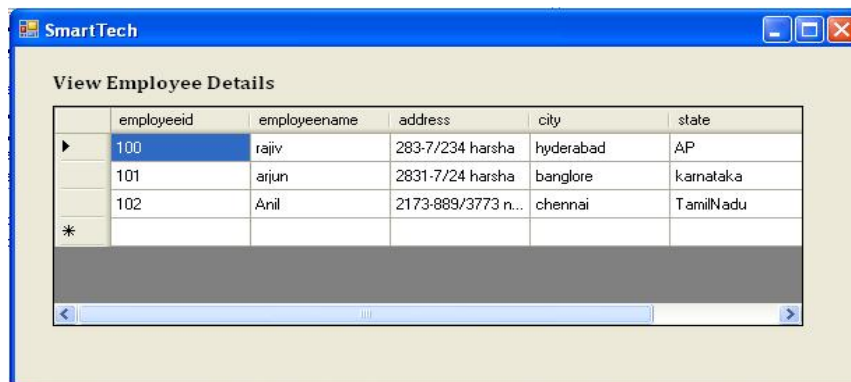


Figure 2.1-1: Output screenshot showing the DataGrid with details of employees

Step 1: In the Visual Studio code page declare the connection object and the connection string variable.

```
SqlConnection MyCon;
string MyConnectionString;
```

Step 2: Define connection string which specifies database driver, location of the database, user id and password. Open the connection.

```
MyConnectionString = "server=hstslc011;database=northwind;uid=sa;password=satyam";
```

Step 3: Pass the connection string to the Connection object and open the connection.

```
MyCon = New SqlConnection(MyConnectionString);
MyCon.Open();
```

Step4: After the required functions are performed, close the connection either implicitly or explicitly.

.....
.....

```
`Explicitly closing the connection for any Exception during database Access
MyCon.Close();
```

```
`Implicitly closing the connection in the finally block (using the dispose
method)
```

```
MyCon.Dispose();
```



Context:

- .Net framework Data Provider is used for establishing connection with database.
- The selection of Data Providers improves the capability and integrity of the application.
- Connection string is used while instantiating a connection.
- Windows authentication is used when there is a domain controller and the application and database are on the same computer
- Mixed mode is used when users have workgroups and they connect from different domains.
- Implicit method of closing the connection is used to ensure that there is no connection leak.



Practice Session:

- Compare the .NET Framework Data Providers for SQL Server, OLE DB and ODBC with each other.
- Mr. Brett heads the Liberal Asset Management group. The work involves preparation of asset declaration documents of its clients. Prepare an application which establishes connection with the client server (may be SQL Server, OLE DB or ODBC) for retrieving the client's asset details.



Check List:

- Need for Data Providers and their types.
- Choosing the most compatible Data Provider.
- Database Security in connection establishment and its various modes
- Connection string and its format for different data sources.
- Opening and closing connection and its ways



Common Error:

- Incorrect information being provided in the connection string or query string.



Exceptions:

- If errors in syntax of the connection string are found, an Argument Exception is thrown.
- An Exception saying that there is an existing open connection, may occur in case of explicit closing of connection if we don't ensure that the close() method is specified in the Finally block. This occurs when the close () method is specified in Catch block (during code's successful execution) or when it is put in the Try block itself (during unsuccessful execution of code).



Lessons Learnt:

- ☑ When operations have been concluded over an open Connection object, the Close method should be invoked on the Connection object to free any associated system resources.
- ☑ Closing a Connection object does not remove it from memory; its property settings can be changed allowing the opening of the connection later on.
- ☑ To completely eliminate an object from memory, set the Connection object variable to **nothing**.



Best Practices:

- Keep a connection open for as short a period as possible
- Connection strings should not be hard-coded in the application. It is better to include them in a configuration file that can be updated easily. Application will have to be re-compiled if its containing connection string will ever have to be changed.
- Password should be encrypted if it is required in an external file.



Topic: Handling Connection Exceptions

Estimated Time: 50 mins.



Objectives: At the end of the session, the participant will be able to:

- Understand methods of handling different types of Exceptions.
- Appreciate the difference in the utility of generic Exceptions and Exceptions belonging to System. Exception.



Presentation:

Try-Catch-finally statements

- It contains three main blocks:
 - Try – contains the real programming code to be controlled for any possible error.
 - Catch - contains the code to handle any error that occurs in try block during program execution.
 - Finally – always executes last, regardless of whether the code in the try block was executed successfully or not.

Multiple Exceptions

- Each try block can have multiple Catch blocks to handle Exceptions of specific type. For e.g. Format Exception, IndexOutOfRangeException, NullReferenceException.
- More specialized catch blocks should come before a generalized one.

Generic Exceptions

- In this type, it is possible that the Exception objects are not subclasses of System. Exception class.
- Under such circumstances, you can omit the Exception type altogether after the catch keyword.



Scenario(Multiple Exceptions):

This example illustrates how to handle multiple Exceptions in one try block .



Demonstration/Code Snippet:

Step 1: Open Visual Studio and create a console application

Step 2: Begin with try and enter the following code. The code below takes two numbers as input and divides them.


```
class Module1
{
    public void Main()
    {
        try {
            int[] array = new int[3];
            double a;
            double b;
            double c;
            object obj;
            Console.WriteLine("Enter 1'st number");
            a = double.Parse(Console.ReadLine);
            Console.WriteLine("Enter 2'nd number");
            b = double.Parse(Console.ReadLine);
            c = a / b;
            Console.WriteLine("Result is " + c);
            array(3) = 1;
            //If the object is not initialized, the line below will cause
            an Exception of type NullReference.
            obj.ToString();
        }

        //If one of a and b is not in numeric format the following catch
        block is executed.
        catch (FormatException Ex) {
            Console.WriteLine("Format - " + Ex.Message);
        }

        //If the array exceeds its range the following catch is executed.
        catch (IndexOutOfRangeException ex) {
            Console.WriteLine("OutBound - " + ex.Message);
        }

        //An un-initialized object causes the following catch block to be
        executed.
        catch (NullReferenceException ex) {
            Console.WriteLine("NullRef " + ex.Message);
        }
    }
}
```



Scenario(Generic Exceptions):

Mr.Thomson owns ZatShoppy, a shopping Portal, and he wants to authenticate his users before using his online portal . The user id contains just the numbers (no characters Allowed). Make a login application which checks userid , raises an Exception if the id is non-numeric and also raises an Exception for Invalid login.



Demonstration/Code Snippet:



Figure 2.4-1: Output Screenshot of the Login Page

Exceptions To be raised

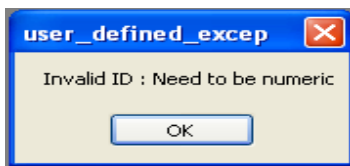


Figure 2.4-2: Output screen showing the numeric Exception occurred

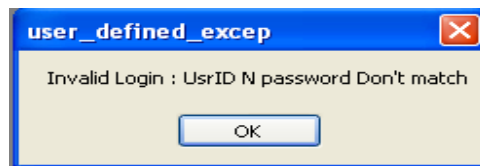


Figure 2.4-3: Output screen showing the invalid login Exception.

There are two ways to handle User-defined Exceptions

Step 1 : Define the MyException Class which inherits the System.Exception and overrides the baseclass.

```
public class MyException : System.Exception
{
    public MyException(string ErrMsg) : base(ErrMsg)
    {
    }
}
```

Step 2 : Define the MyException2 Class which inherits the System.Exception.

```
public class MyException2 : System.Exception
{
    public string InvalidLogin;
}
```

Step3 : Create an instance to the MyException2 class assigning the error message to be displayed.

```
public class Generic_Exception
{
    private void Login_Click(object sender, System.EventArgs e)
    {
        MyException2 expobj = new MyException2();
        expobj.InvalidLogin = "Invalid Login : UsrID N password Don't match";
        try {
            if (Information.IsNumeric(usrname.Text)) {
                if ((usrname.Text == "100" & pwd.Text == "eltp")) {
                    Interaction.MsgBox("Welcome Eltp");
                }
                else {
                    //The following statements throw a user-defined Exception
                    throw expobj;
                }
            }
            else {
                throw (new MyException("Invalid ID : Need to be numeric"));
            }
        }

        //The following statements catch Exceptions of generic type
        catch (MyException ex) {
            Interaction.MsgBox(ex.Message);
        }
        catch (MyException2 ex) {
            Interaction.MsgBox(ex.InvalidLogin);
        }
    }
}
```



Context:

- To handle unforeseen conditions that might occur during the execution of an application.
- They help in setting the order in the way it is best to handle the code's possible errors.



Practice Session:

- Compare the advantages of Exception handling over error notification methods.
- Mr. Matthew is making a new E-mail offering over the internet. Prepare an application concerning "New User Registration Form" containing fields like First Name (string), Last Name (string), Gender (char), Birth date (numeric), Nationality (string), E-mail ID (alphanumeric), Password (alphanumeric), Alternate E-mail ID, Security Question and Answer (alphanumeric). The application should handle all the possible errors in data entry using Exception handling mechanism.



Check List:

- Implementation of Try-Catch-Finally statements.
- Handling multiple and generic Exceptions.



Common Errors:

- The code, in which there is a possibility of an Exception being raised, is kept out of the try block.
- The “new” keyword is missed during instantiating an object which results in the constructor of that object not being initialized.



Exception:

- Error notification methods may be employed over Exception handling in applications where the expected errors would not be fatal for the application or system, as these methods do not use system resources.



Lessons Learnt:

- ☑ The runtime Exception handling is faster than Windows-based error handling.
- ☑ If there is no specific catch block, the Exception is caught by a general catch block, if one exists.
- ☑ The common language runtime find out Exceptions that are not identified by a catch block.
- ☑ The intermediate results should be cleaned up when throwing an Exception. Callers should be able to assume that there are no side effects when an Exception is thrown from a method.



Best Practices:

- Use try/finally blocks around a code that can potentially generate an Exception and centralize the catch statements in one location.
- End Exception class names with the word "Exception". For example
 - `public class MyFileNotFoundException : ApplicationException`
- In most cases, use the predefined Exceptions types.
- Throw Exceptions instead of returning an error code.
- It is also good practice to implement the three recommended common constructors
 - `public EmployeeListNotFoundException()`
 - `public EmployeeListNotFoundException(string message) : base(message)`
 - `public EmployeeListNotFoundException(string message, Exception inner): base(message, inner)`
- Exceptions handling is quite costly. So avoid handling Exceptions for those which can be handled using if-else statements.



Topic: Connection Pooling

Estimated Time: 20 mins.



Objectives: At the end of the session, the participant will be able to:

- Create a pool of connections.
- Work with multiple connections at a time.



Presentation:

- Connection pooling allows reusing connections rather than creating a new one every time the ADO.NET data provider needs to establish a connection to the underlying database.
- Each connection pool is associated with a specific connection string.
- The pool is populated with connections up to the minimum pool size.



Scenario:

In the following C# code fragment, three new SqlConnection objects are created, but only two connection pools are required to manage them. Note that the connection strings for conn1 and conn2 differ by the values assigned for User ID, Password, and Min Pool Size specified in it.



Demonstration/Code Snippet:

'Specification in the connection string: Connection Reset=False; Connection Lifetime=5; Enlist=true; Min Pool Size=0Max Pool Size= 50. Pool A is created for conn1 as it is a new connection.

```
{
    SqlConnection con1 = new SqlConnection();
    con1.ConnectionString = "server=hstslc011;database=northwind;" +
        "uid=sa;password=satyam;connection reset=false;connection lifetime=5;" +
        "enlist=true;min pool size=0;max pool size=50";
    con1.Open();

    //Pool B is created because the connection string con2 differs in User
    ID, Password, and Min Pool Size with con1.

    SqlConnection con2 = new SqlConnection();
    Con2.ConnectionString = "server=hstslc011;database=northwind;" +
        "uid=eltp;password=satyam;connection reset=false;connection lifetime=5;" +
        "enlist=true;min pool size=5;max pool size=50";
    Con2.Open();

    //Con3 is assigned an existing Connection Pool A as the connections
    strings for both are same.

    SqlConnection con3 = new SqlConnection();
    Con3.ConnectionString = "server=hstslc011;database=northwind;" +
        "uid=sa;password=satyam;connection reset=false;connection lifetime=5;" +
        "enlist=true;min pool size=0;max pool size=50";
```

```
Con3.Open();  
  
//Return all connections to their specific pools.  
con1.Close();  
con2.Close();  
con3.Close();  
}
```



Context:

- Connection pooling delivers an optimal environment for database-driven applications.
- It is used for reusability of connections.



Practice Session:

- Explore and enlist the problems resulting out of pool fragmentation due to Integrated Security and multiple databases on a server.
- In the above code snippet, two connection pools have been created. Suggest a way through which only one connection pool will be created catering to all the three connection strings.



Check List:

- Need for connection pooling.
- Implementation of connection pooling.



Common Errors:

- A huge number of connection pools being left open causing pool fragmentation and degradation of system's performance.
- Connection pooling might not be supported by the framework that is being used.



Exceptions:

- If the connection pool has exceeded its limit and is unable to reclaim any existing connection for accommodating within a specific period of time called connection time-out, it gives an Exception.
- If you are using .net compact framework, some SqlConnection.ConnectionString properties related to connection pooling might not be supported, resulting in an InvalidOperationException Exception being thrown.



Lessons Learnt:

- ☑ Pooling connections can significantly enhance the performance and scalability of your application.
- ☑ A connection that has gone out of scope but that has not been explicitly closed will only be returned to the connection pool if the maximum pool size has been reached and the connection

is still valid.

- ☑ A connection pool is created for each unique connection string.
- ☑ When a connection is closed, it is released back into the pool and into the appropriate subdivision based on its transaction context. Therefore, you can close the connection without generating an error, even though a distributed transaction is still pending.

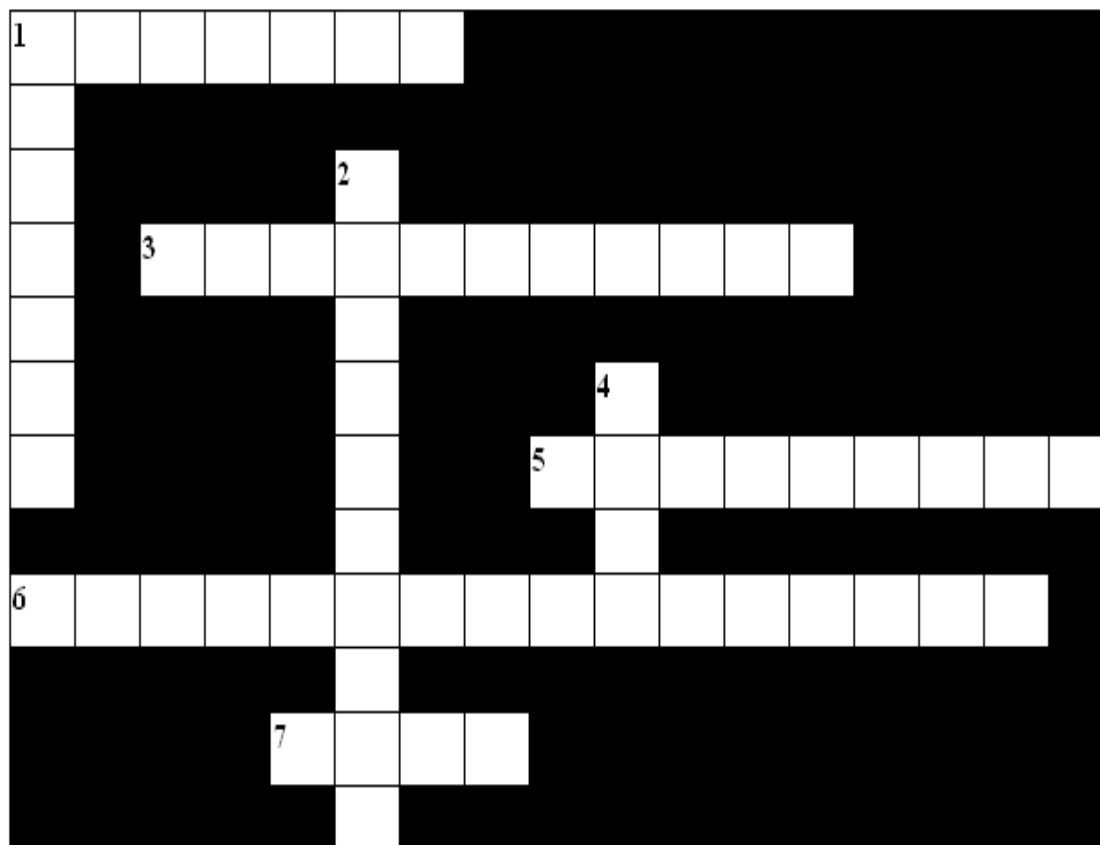


Best Practices:

- It is always recommended that the connection is closed or disposed soon after its use, in order to return it to the pool.
- Do not call the **Close** or **Dispose** on a **Connection**, a **DataReader**, or any other managed object in the **Finalize** method of a class. In a finalizer, only unmanaged resources which are owned directly by the class are to be released. . If the class does not own any unmanaged resources, do not include a **Finalize ()** method in the class definition.

Crossword: Unit-2

Estimated Time: 10 mins.



Across:











1. The method used to close the connection with the database in dot net (7).
3. Which component of ADO.NET exists between the local repository and the physical database (11).
5. The best place to store connection string in .NET projects.(9)
6. The class used in ADO.NET which makes a code block transactional, which cannot be inherited(16)
7. When we do not close/dispose the connection, GC collects them in its own time; such connections are considered as (4).

Down:

1. The local repository of the data used to store the tables and disconnected record set is known as (7).
2. Component of ADO.NET reads in forward-only direction uses connected architecture and uses read-only stream of data from a data source (10).
4. The default Life time of a connection (In minutes) (4).

3.0 Performing Connected Database Operations

Topics

-  3.1 Working in a Connected Environment
-  3.2 Building Command Objects
-  3.3 Executing Command Objects that return single value
-  3.4 Executing Commands that return rows
-  3.5 Executing Commands that do not return rows
-  3.6 Command Parameters
-  3.7 Executing Stored Procedures
-  3.8 Using Transactions
-  3.9 Transaction Vocabulary
-  3.10 Crossword





Topic: Working in a Connected Environment

Estimated Time: 50 mins.



Objectives : On completion of the activity, the participant will be able to:

- Connect to data source and retrieve scalar value
- Build Command objects
- Execute Command object which
 - returns a single value
 - returns multiple rows
 - inserts/updates/deletes and returns no rows



Presentation:

- Connection object represents a connection to the data source.
- Connection object facilitates other objects like the Data Adapter and the Command objects to communicate with the database submit queries and retrieve results.
- A scalar value can be retrieved in two ways:
 - Using DataSet when working with the same records repeatedly.
 - Using a data command when performing a non-query operation, such as a DDL command.

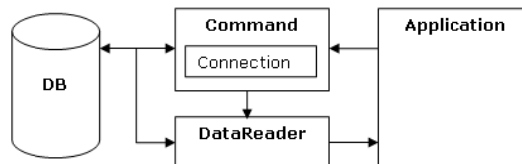


Figure 3.1-1: Connected Architecture

Building Command objects

- Command object is represented by the **SqlCommand** class.
- Three methods of executing commands on the database are:
 - **ExecuteReader**. Returns a result set by way of a DataReader object.
 - **ExecuteNonQuery**. Executes direct SQL commands, such as INSERT, UPDATE or DELETE.
 - **ExecuteScalar**. Returns a single value from a Database Query

Executing Command Object that Returns a single value

- Call ExecuteScalar () over a SQL statement or stored procedure that returns a single value after setting its CommandText property.

- ExecuteScalar method returns an Object, so you must cast the ExecuteScalar method to the equivalent of the returned data type.

Executing Command Object that Returns Rows

- ExecuteReader () returns multiple rows.
- To read the results, create a DataReader object that holds the data returned from the database.

Executing Command that inserts/updates/deletes without returning rows

- ExecuteNonQuery () executes queries without returning any rows.
- Define the parameters and assign values to them.
- The ExecuteNonQuery () automatically updates, inserts or deletes as per the query passed to the Command object.



Scenario:

Mr. Hoops owns PowerElectricals Hardware company . He wants to view the details of all the employees and add new records to the database. The employeeinfo table contains eid, ename, dob, doj, contactno, mailid . Create an application which allows the user to read details and add new records into the database.



Demonstration/Code Snippet:

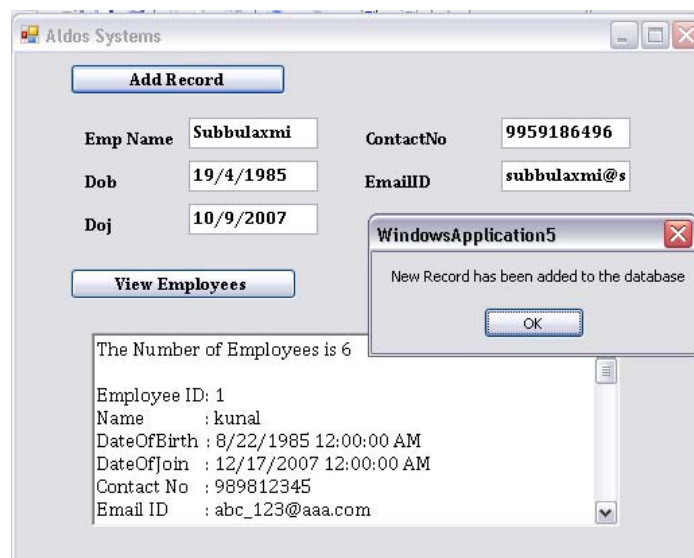


Figure 3.1-2: output screenshot showing the added records to the database

Step 1: Open visual studio and create a new windows application.

Step 2: Import the following namespaces.

```
using System.Data;  
using system.data.sqlclient;
```

Step 3: Define the Employees class.

```
public class EmployeesDB  
{  
    SqlConnection DbCon = new SqlConnection();  
}
```

Step 4: This load method is used for defining the connection String.

```
private void EmployeesDB_Load(object sender, System.EventArgs e)  
{  
    string strConstring =  
        "server=hstslc011;database=northwind;uid=sa;password=satyam";  
    DbCon = new SqlConnection(strConstring);  
}
```

Step 5 : ViewEmployees_Dept Method is use to display the details of all the employees present in the company.

```
private void ViewEmployees_Dept_Click(object sender, System.EventArgs e)  
{  
    try {  
        // open connection  
        DbCon.Open();  
  
        //EXECUTESCALAR COMMAND  
        // Create a command object and set the commandtype and query to be  
        // executed  
  
        SqlCommand ScalarCommand = new SqlCommand();  
        ScalarCommand.CommandType = CommandType.Text;  
        ScalarCommand.CommandText = "select Count(*) from employeeinfo";  
        // Assign the connection to the command object  
        ScalarCommand.Connection = DbCon;  
        // Execute the ExecuteScalar method and typecast the result  
        // to a specific DataType for assigning it to a variable  
        int EmpCount = (int)ScalarCommand.ExecuteScalar;  
        Interaction.MsgBox(EmpCount);  
        // EXECUTEREADER COMMAND  
        // Instantiate the command Object and set the commandtype and query to  
        // be executed  
        SqlCommand ReaderCommand = new SqlCommand();  
        ReaderCommand.CommandType = CommandType.Text;  
        ReaderCommand.CommandText = "select * from employeeinfo";  
        // Assign the connection to the command object
```

```

ReaderCommand.Connection = DbCon;
// Execute the ExecuteReader method and assign the output to
DataReader
SqlDataReader dr = ReaderCommand.ExecuteReader;
EmpsDisplay.Text = "The Number of Employees is " + EmpCount;

// Read the data record-by-record using while.
// Here dr.read is used to navigate to the next record
while (dr.Read) {
    EmpsDisplay.Text += "Employee ID: " + dr.GetValue(0).ToString
    EmpsDisplay.Text += "Name          : " + dr.GetValue(1).ToString
    EmpsDisplay.Text += "DateOfBirth: " + dr.GetValue(2).ToString
    EmpsDisplay.Text += "DateOfJoin  : " + dr.GetValue(3).ToString
    EmpsDisplay.Text += "Contact No : " + dr.GetValue(4).ToString
    EmpsDisplay.Text += "Email ID   : " + dr.GetValue(5).ToString

}
// Close the datareader
dr.Close();
}
catch (Exception ex) {
    MsgBox.show(ex.Message);
}
finally {
    //Close the connection
    DbCon.Close();
}
}

```

Step 6: AddRecord Method is used to add the details of a new employee

```
private void AddRecord_Click(object sender, System.EventArgs e)
```

```

    try {
        DbCon.Open();
        // EXECUTENONQUERY(INSERT COMMAND)
        // Create the command Object, assign the connection object
        and set the commandtype.
        SqlCommand InsertCommand = new SqlCommand();
        InsertCommand.Connection = DbCon;
        InsertCommand.CommandType = CommandType.Text;
        // Assign the query to be executed
        InsertCommand.CommandText = "insert into employeeinfo
values (@ename,@dob,@doj,@cntctno,@mailid)";
        // Define the parameters specified in the query like
        parameter name,type, length etc and assign values to them
        InsertCommand.Parameters.Add("@ename", SqlDbType.NVarChar,
30).Value = EmpName.Text;
        InsertCommand.Parameters.Add("@dob", SqlDbType.NVarChar,
10).Value = Dob.Text;
        InsertCommand.Parameters.Add("@doj", SqlDbType.NVarChar,
10).Value = Doj.Text;
    }
}

```

```

        InsertCommand.Parameters.Add("@cntctno",
SqlDbType.NVarChar, 30).Value = ContactNo.Text;
        InsertCommand.Parameters.Add("@mailid",
SqlDbType.NVarChar, 30).Value = EmailID.Text;
        // Execute the insertcommand using ExecuteNonQuery()
        InsertCommand.ExecuteNonQuery();

        MsgBox.Show("New Record has been added to the database");
    }
    catch (Exception ex) {
        MsgBox.Show(ex.Message);
    }
    finally {
        DbCon.Close();
    }
}

```

Table Used

DataBase: NorthWind ; Table: employeeinfo


	Column Name	Data Type	Length	Allow Nulls
	eid	int	4	
	ename	varchar	30	
	dob	datetime	8	
	doj	datetime	8	
	cntctno	varchar	12	
	mailid	varchar	30	

Figure 3.1-3: Table used for the above scenario



Context:

- Command objects are used for executing statements (commands) and stored procedures to a database.
- ExecuteScalar is most useful when a single value is to be retrieved with less coding (ExecuteReader would require more code).



Practice Session:

- Amazon motor works is a leading enterprise dealing in automobile parts. The administration of their web application deals with adding details of new product ranges and extracting details of existing ranges from the products table. Prepare a windows-based application regarding the same containing fields like product ID, product Name, Product Cost, Size Specifications and Material Specifications.



Check List:

- Defining a command object.

- Methods provided in the Command object.
- Ways of retrieving a single value or a complete row from the database.
- Executing a command without returning a row.



Common Errors:

- Incorrect information provided in the connection string or query string.
- Command object has not been assigned with connection before executing the query.
- Incorrect details specified while adding parameters to the command object.
- Not closing the data reader or connection object.
- Using ExecuteQuery instead of ExecuteNonQuery.



Exceptions:

- IOException might occur if an I/O error occurred while reading from the input stream or writing to the output stream.
- OutOfMemory Exception might occur if there is insufficient memory for executing the program.
- ArgumentException might occur if the values specified in any argument are invalid.



Lessons Learnt:

- ✓ Building a Command object.
- ✓ Returning a single value using ExecuteScalar method in the command object
- ✓ Returning a row using ExecuteReader method in the command object
- ✓ ExecuteNonQuery Method is used to execute the command without returning anything.



Best Practices:

- The **DataReader** must be closed before accessing any output parameters for the associated **Command**.
- Passing **CommandBehavior.CloseConnection** to the **ExecuteReader** method ensures that the associated connection is closed when the **DataReader** is closed. This is especially useful when returning a **DataReader** from a method without having control over the closing of the **DataReader** or associated connection.
- Use typed accessors like **GetString**, **GetInt32**, and so on to access column data which saves the processing required to cast the **Object** returned from **GetValue** as a particular type.
- Passing **CommandBehavior.SequentialAccess** to the call to **ExecuteReader** changes the default behavior of the **DataReader** to only load data into memory when it is requested.



Topic: Working with Command Parameters and Stored Procedures

Estimated Time: 40 mins.



Objectives: At the end of the session, the participant will be able to:

- Create and add parameters to Command object
- Execute Stored Procedures



Presentation:

Command Parameters

- A parameter passes and returns values between an application and a database.
- Three types of parameters are:
 - **Input parameter:** sends data to the database.
 - **Output parameter:** retrieves information from the database.
 - **InputOutput parameters:** send as well as receive data

Stored Procedures

- Stored procedures perform database INSERT, UPDATE, and DELETE operations and retrieve single values and result sets.
- Stored procedures give performance benefits and restrict data access to the predefined interfaces that they expose.
- Default direction of all parameters passed to a stored procedure is input.



Scenario:

Mr. Wilson is the company manager of KWIL Insurance services. He wants to have a login screen in windows form. Wilson decided to use stored procedures and command parameters to make the code reusable and provide more security to the application.



Demonstration/Code Snippet:

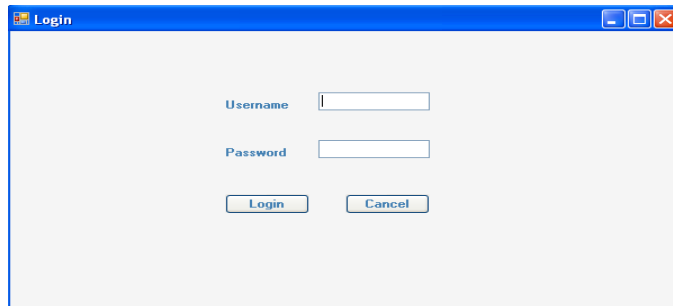


Figure 3.6-1: Login Page of KWIL Insurance Services

Step 1: Open the query analyzer and create the following Stored Procedure.

```
create proc loginprocess @eid int,@pwd varchar(30),@reply int output
as
begin
select @reply=count(*) from emplogindetails where eid=@eid and pwd=@pwd
end
```

Step 2: Import the necessary namespaces, and declare the connection and command object.

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

Step 3: Defining Login Class

```
public class login
{
    SqlConnection con;
    SqlCommand cmd;

    //Define the connection string
    String connectionstr =
"server=hstslc011;uid=sa;pwd=satyam;database=northwind";
}
```

Step 4: Enter the following code in the Page Load event.

```
private void login_Load(object sender, System.EventArgs e)
{

    txt_username.Text = "";
    txt_password.Text = "";
    txt_username.Focus();

}
```

Step 5: Add the following code to cancel button click event

```
private void btn_cancel_Click(object sender, System.EventArgs e)
{

    txt_username.Text = "";
    txt_password.Text = "";
    txt_username.Focus();

}
```

Step 6: Validate whether the required information has been provided or not in the login button click event.



```
private void btn_login_Click(object sender, System.EventArgs e)
{
    if ((txt_username.Text == "")) {
        Interaction.MsgBox("Please Enter Username", MsgBoxStyle.Information);
        txt_username.Focus();
    }
    else if ((txt_password.Text == "")) {
        Interaction.MsgBox("Please Enter Password", MsgBoxStyle.Information);
        txt_password.Focus();
    }
    else {
        //Establish connection with the database using stored procedures
        try {
            con = new SqlConnection(connectionstr);
            cmd = new SqlCommand("loginprocess", con);
            //To run a stored procedure, we need to set the command type
            property
            cmd.CommandType = CommandType.StoredProcedure;
            //Add parameters to stored procedure(names should exactly match)
            and set the direction property
            cmd.Parameters.Add("@eid",          SqlDbType.Int).Value          =
            (int)txt_username.Text;
            cmd.Parameters.Add("@pwd",          SqlDbType.VarChar,          30).Value          =
            txt_password.Text;
            cmd.Parameters.Add("@reply",        SqlDbType.Int).Direction      =
            parameterDirection.Output;
            //Open the connection
            con.Open();
            //Execute the command using ExecuteNonQuery method
            cmd.ExecuteNonQuery();
        }
        catch (Exception ex) {
            Interaction.MsgBox("Login Error", MsgBoxStyle.Information);
        }

        //Close the connection with the database
        finally {
            con.Close();
        }

        //Authenticate the provided login details and provide further access
        if (cmd.Parameters("@reply").Value > 0) {
            homepage.Show();
        }
        else {
            Interaction.MsgBox("Login Failed", MsgBoxStyle.Critical, "Login
Error");
        }
    }
}
```

Tables used: emplogindetails ; **Database:** northwind

Table - dbo.emplogindetails		hstslc011.Nort...SQLQuery1.sql*
Column Name	Data Type	Allow Nulls
eid	int	<input checked="" type="checkbox"/>
pwd	varchar(30)	<input checked="" type="checkbox"/>
		<input type="checkbox"/>

Figure 3.6-2: Table used in the above scenario



Context:

- Command Parameters allow values to be passed to the data source at run-time when parameters are defined before.
- Stored procedures are pre-compiled, thereby improving the application's performance and optimizing the system resources as now the typically required parsing steps and the need to create an access plan is not required.
- Network traffic is reduced because the set of statements to be sent are reduced.



Practice Session:

- Hudson Online Bookkeeping Services deals with doing accounting work for its clients. The clients are required to log into the system with proper login credentials to view their account documents. Prepare an application using stored procedures concerning the relevant login form. The MemberLogin table contains fields like LoginID and password. The following page displays the information of the Members table like MemberID, Member Name and Client Profile.



Check List:

- Creating Command Parameters
- Adding Parameters to Command Objects
- Creating and executing Stored Procedures.



Common Errors:

- Connection not opened before Execution of the query
- Incorrect information provided in the connection string.
- Not specifying the correct name, data-type, value, size and direction while adding parameters to the command object.



Exceptions:

- SQL Exception might occur if the stored procedure is not found
 - Argument Exception might occur if the details specified while passing parameters to the DataSource do not match with its columns.



Lessons Learnt:

- ☑ Command Parameters and their types
- ☑ Purpose of the various parameters
- ☑ How to create a stored procedure



Best Practices:

- Specify a **CommandType** of **StoredProcedure** for the **CommandType** property of the **SQLCommand** for calling a stored procedure. This removes the need to parse the command before execution.



Topic: Transactions and Transaction Vocabulary

Estimated Time: 50 mins.



Objectives: At the end of the activity, the participant will be able to:

- Understand the need for transactions.
- Manage transactions.



Presentation:

Transaction Vocabulary

- Transactions are a group of database commands that execute as a package.
- Transactions help the Database in satisfying ACID properties.
- A transaction enabled on a single connection to the database applies to all commands executed over that connection
- Transactions in ADO.Net are performed by SQLTransaction class that belongs to System.Data.SqlClient namespace.
- Three important commands to manage a transaction are:
 - Begin Transaction(): Starting Point of Transaction
 - CommitTransaction (): It ends the transaction successfully if no errors are encountered.
 - Rollback(): It erases the uncommitted transactional steps in which errors are encountered.
 - Save (): It creates a savepoint in the transaction.
- Types of transactions:
 - Local
 - Distributed
 - Manual
 - Automatic



Scenario:

Mr. Anderson works as a systems manager with Deutsche International which is a world renowned enterprise in providing financial services to customers. The firm has decided to get its system automated using a software which maintains consistency with the customer's accounts while performing transfer, debit and credit operations over it. The work involves the development of a windows application which performs transactions like transferring money from one account to another.



Demonstration/Code Snippet:



Figure 3.8-Output screen showing a successful money transfer

Step 1: Import Libraries for access to SQLServer

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

```
public class Bank
{
```

Step 2: This method entails checking and displaying the customer's balance. Begin with defining the connection string and opening the connection.

```
private void btnChkBalance_Click(object sender, System.EventArgs e)
{
    SqlConnection con_north = new
    SqlConnection("server=hstslc011;database=northwind;uid=sa;password=satyam");
    con_north.Open();
    try {
        string balance;
        //Retrieve the balance amount from the database.
        balance = string.Format("select cust_balance from deutsche_customers
where cust_account={0}", Conversion.Val(txtFromAccount.Text));
        SqlCommand getBalCommand = new SqlCommand(balance, con_north);
        int bal;
        //Executescalar returns the value in the result's 1st column
        bal = getBalCommand.ExecuteScalar;
        txtBalance.Text = bal;
    }
    catch (Exception ex) {
        //catches the Exception
        MessageBox.Show(ex.message);
    }
}
```

```

    }
    finally {
        con_north.Close();
        //closes the connection with the datasource
    }
}

```

Step 3: The method below involves creation of a new account. It deals with a single database(northwind in our example). Begin with defining the connection string and opening the connection.

```

private void btnNewCust_Click(object sender, System.EventArgs e)
{
    SqlConnection con_north = new
    SqlConnection("server=hstslc011;database=northwind;uid=sa;password=satyam");
    Con_north.Open();

    //Start a local explicit(manual) transaction by using the BeginTransaction
    method of SqlTransaction class.
    SqlTransaction myTrans = con_north.BeginTransaction();
    try {

        //Enlist the command in the current transaction.
        string newcust;
        newcust = string.Format("Insert into deutsche_customers
(cust_name,cust_balance,cust_account) VALUES ({0},{1},{2})",
Conversion.Val(txtName.Text), Conversion.Val(txtBalance.Text),
Conversion.Val(txtFromAccount.Text));

        SqlCommand myCommand = new SqlCommand(newcust, con_north);
        myCommand.Transaction = myTrans;
        myCommand.ExecuteNonQuery();
        //returns the number of rows affected after executing the query

        //Commit the transaction to the database permanently if it executes
        successfully.
        myTrans.Commit();
        Interaction.MsgBox("New account created");
    }
    catch (Exception ex) {

        //If the transaction fails, undo all the uncommitted transactional
        changes by using the rollback operation and close the open database
        connection.
        myTrans.Rollback();
        MessageBox.Show(ex.message);
    }
    finally {
        con_north.Close();
    }
}

```



Step 4: This method deals with transferring money from one account to another which may span across more than one database(like here) or even across more than one data source. Begin with defining connection string and opening the connection.

```
private void btnTransfer_Click(object sender, System.EventArgs e)
{

    SqlConnection con_north = new
    SqlConnection("server=hstslc011;database=northwind;uid=sa;password=satyam");
    con_north.Open();

    SqlConnection con_deut = new
    SqlConnection("server=hstslc011;database=deutsche;uid=sa;password=satyam");
    Con_deut.Open();

    //Start a distributed explicit(manual) transaction by using the
    BeginTransaction method.
    SqlTransaction trans_north = con_north.BeginTransaction();
    SqlTransaction trans_deut = con_deut.BeginTransaction();
    try {

        //Enlist the commands in their transactions.

        string transfrom;
        transfrom = string.Format("update deutsche_customers set
cust_balance=cust_balance-{0} where cust_account={1}",
Conversion.Val(txtTransfer.Text), Conversion.Val(txtFromAccount.Text));

        SqlCommand com_north = new SqlCommand(transfrom, con_north);
        Com_north.Transaction = tran_north;

        string transto;
        transto = string.Format("update Deustche_Customers set
cust_balance=cust_balance+{0} where cust_account={1}",
Conversion.Val(txtTransfer.Text), Conversion.Val(txtToAccount.Text));

        SqlCommand com_deut = new SqlCommand(transto, con_deut);
        Com_deut.Transaction = trans_deut;

        //Commit the transactions to the database by using the commit method
        only if they execute successfully.
        int a;
        int b;
        a = com_north.ExecuteNonQuery();
        b = com_deut.ExecuteNonQuery();
        if (a > 0 & b > 0) {
            Trans_north.Commit();
            Trans_deut.Commit();

            //Display the remaining balance
            string balance;
            balance = string.Format("select cust_balance from
```



```

deutsche_customers where cust_account={0}",
Conversion.Val(txtFromAccount.Text));
    SqlCommand getBalCommand = new SqlCommand(balance, con_north);
    int bal;
    bal = getBalCommand.ExecuteScalar;
    string exbal;
    exbal = string.Format("Transfer Done.....Remaining balance={0}",
bal);
    Interaction.MsgBox(exbal);
}
}
catch (Exception ex) {

    //If the transactions fail, undo all the uncommitted transactional
changes by using the rollback operation and close the open database
connections.
    Trans_north.Rollback();
    Trans_deut.Rollback();
    Interaction.MsgBox(ex.Message);
}
finally {
    con_north.Close();
    con_north1.Close();
}
}

```

Tables used:

Two tables are used which are similar in terms of structural definition with the only difference being that they belong to different databases...northwind and deutsche(custom-made).

Table - dbo.De...tche_Customers		Table - dbo.deutsche_customers	
Column Name	Data Type	Allow Nulls	
Cust_Name	varchar(50)	<input type="checkbox"/>	
Cust_Balance	int	<input type="checkbox"/>	
Cust_Account	varchar(50)	<input type="checkbox"/>	
		<input type="checkbox"/>	

Figure 3.8-Tables used in the above scenario



Context:

- Transactions are used when two commands are to be executed in such a way that if one fails, the other should also not be able to modify the database.
- Transactions allow applications to abort (roll back) all changes executed within the transaction if any errors occur during the transaction process.



Practice Session:

- McCain Retail is a leading retail chain. They have set up zonal level warehouses for stocking their goods. Their web module related with warehouse management requires dynamic

reflection of their warehouse's status on their website so that the requirements of their outlet are consistently in line with the available goods. Functionality dealing with supply of goods by supplier and reception of goods by warehouse forms one transaction and another transaction consists of release of goods by warehouse and reception of goods by outlets. Prepare an application for the same using transactions.

- Explore the System. Transactions namespace and build the above application using implicit transactions.



Check list:

- Importance of transactions
- Commands used to manage transactions.



Common Errors:

- Incorrect information being specified in the connection string or query string.
- Commands not being enlisted in the ongoing transaction.
- Setting the Command. Transaction property to a transaction which has not been started.



Exceptions:

- Argument Exception might occur during rollback if a proper saved state is not found.
- Invalid Operation Exception might occur if the transaction is altered in such a way that its isolation level is altered.



Lessons Learnt:

- Properties and Methods involved in SqlTransaction class
- Steps to implement a transaction.
- Overview of the ACID properties and their importance in transactions.

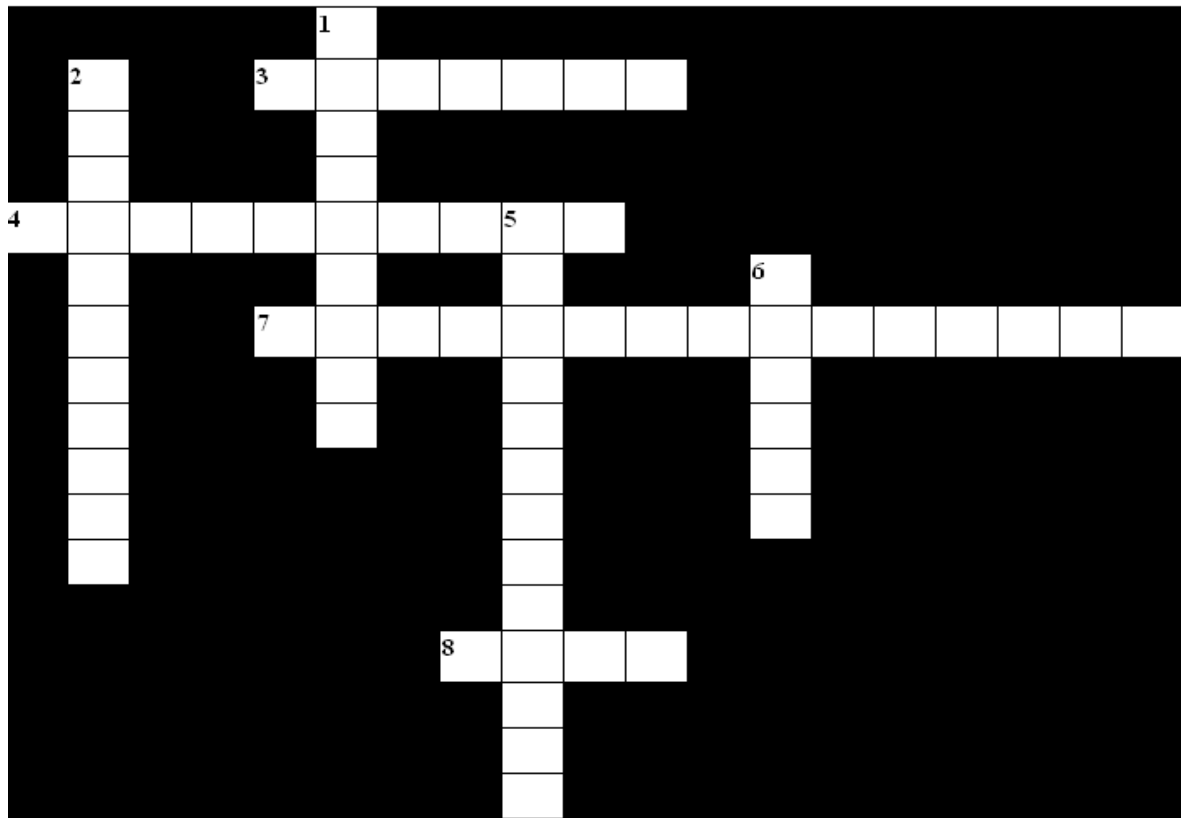


Best Practices:

- When Connection.BeginTransaction is called, a Transaction object is returned that needs to be associated with the Transaction property of a Command. This design enables you to perform multiple root transactions out of a single connection.

Crossword: Unit-3

Estimated Time: 10 mins.



Across:









2. The important objects for working in the connected environment ()
3. a component that reads the data from the database management system and provides it to the application
6. a precompiled executable object that contains one or more SQL statements
7. This save point can be used to rollback (4).

Down:

1. Connections are needed to open and close manually.
3. A class that connects to the database system fetches the record and fills the DataSet. It contains four different commands to perform database operations; Select, Update, Insert, Delete.
4. To execute commands that return single values
5. the methods for transaction

4.0 Building DataSets

Topics

-  4.1 Working in a Disconnected Environment
-  4.2 DataSet Object Model
-  4.3 Building DataSets and Data Tables
-  4.4 Binding and Saving a DataSet
-  4.5 Defining Data Relationships
-  4.6 Modifying Data in a Data Table
-  4.7 Sorting and Filtering
-  4.8 Crossword





Topic: Disconnected Environment and DataSet Object Model

Estimated Time: 50 mins.



Objectives: At the end of the session, the participant will be able to understand:

- Data access with Disconnected Architecture
- Objects involved in DataSet model.
- Building DataSet and DataTable.



Presentation:

The DataSet is the core component of the Disconnected Architecture of ADO.NET. The **DataSet** is explicitly designed for data access independent of any data source.

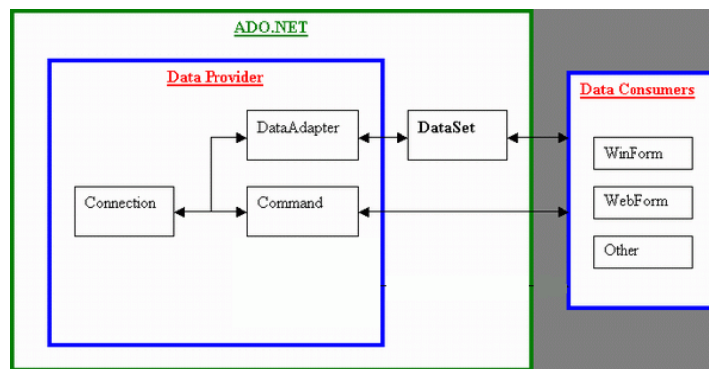


Figure 4.1-1: Disconnected Object Model

Disconnected Architecture

- DataSet object maintains a local repository of data.
- Application connects to the Database Server when it needs to pass some query and then disconnects immediately after the result set is retrieved and stored in DataSet.
- DataSet object stores the tables, their relations and its constraints.
- The update, insert, delete operations are done to the DataSet locally and finally the changed DataSet is updated to actual database as a batch.

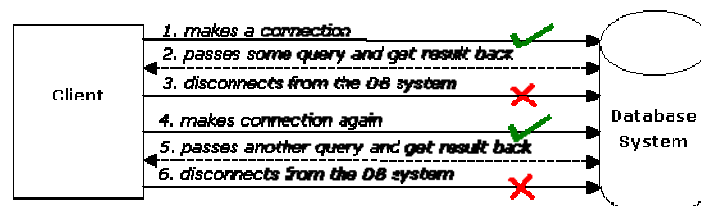


Figure 4.1-2: Disconnected Data Access architecture

DataSet Object Model

- DataSet object is central to supporting disconnected, distributed data scenarios.
- DataSet represents a subset of the entire database, cached on the machine without direct connection to the database.
- DataSet is a container for a number of DataTable objects that are stored in the tables collection.
- The nature of a DataSet is either Typed or Un-Typed
 - Typed DataSet is a DataSet that is derived from the base DataSet class that applies the information contained in the XSD to generate a Typed class.
 - Un-Typed DataSet has no corresponding schema and is exposed only as a collection.

DataTable

- DataTable can contain multiple **DataRows**. Each row contains multiple fields representing each column in the table.
- **DataColumn** is the fundamental building block for creating the schema of a DataTable.
- **Constraints** are applied to a column or related columns, that determine the course of action when the value of a row is altered. Two types of constraints are:
 - ForeignKey Constraint
 - Unique Constraint
- **Data Relation** relates two DataTable objects to each other through DataColumn objects.



Scenario:

Mr.Headlee owns the Peers Training institute and wants the users to view the course details offered by the institute one course at a time. The course details include Course Name, Duration and Fees. Build an application by implementing DataTable and DataRow objects.



Demonstration/Code Snippet:



Fig 4.1-3: Application to show the course Details

Step 1: Open visual studio and create a new windows application

Step 2: Import the following namespaces

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

Step 3: Defining the Course Details class

```
public class Course_Details  
{  
  
    SqlConnection objConn;  
    SqlDataAdapter CourseAdapter;  
    // Create DataSet, DataTable and DataRow object  
    DataSet CourseDataSet = new DataSet();  
    DataTable CourseDataTable = new DataTable();  
    DataRow CourseDataRow;  
    int Row_number;  
}
```

Step 4: Course_details Method is to fill the data from the SqlServer into the CourseDataSet using DataAdapter and finally assigning it to the CourseDataTable

```
private void Course_Details_Load(object sender, System.EventArgs e)  
{  
    try {  
  
        // Create a Connection to the Server  
  
        objConn = new SqlConnection();  
        objConn.ConnectionString =  
"server=htslc011;uid=sa;password=satyam;database=northwind";  
        objConn.Open();  
  
        // Fill the DataSet with the Course table data using DataAdapter  
  
        CourseAdapter = new SqlDataAdapter("SELECT * FROM course", objConn);  
        Row_number = 0;  
        CourseAdapter.Fill(CourseDataSet);  
  
        // Assign a specific table in the DataSet to DataTable(dt) Object  
        CourseDataTable = CourseDataSet.Tables(0);  
  
        // This method is to show details Explained Later  
        show_details();  
    }  
    catch (Exception ex) {  
        MessageBox.Show("FormLoad Exception : " + ex.Message);  
    }  
}
```

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

```
}
```

Step 5: Next_record Method to view the next record

```
private void Next_Record_Click(object sender, System.EventArgs e)  
{  
    if ((Row_number < dt.Rows.Count - 1)) {  
        // Increasing the row number if user wants to view next record  
        Row_number += 1;  
        show_details();  
    }  
    else {  
        MessageBox.Show("No more records left");  
    }  
}
```

```
}
```

Step 6: Previous_record Method to view the previous record

```
private void Previous_record_Click(object sender, System.EventArgs e)  
{  
    if ((Row_number > 0)) {  
        // Decreasing the row number if user wants to view previous record  
        Row_number -= 1;  
        show_details();  
    }  
    else {  
        MessageBox.Show("You are at First Record");  
    }  
}
```

```
}
```

Step 7: Show Details method is to show the details from a specific row

```
Private Sub show_details()  
{  
  
    ' Assign an individual row to the DataRow object based on row number  
    and access the elements of this row  
    try  
    {  
        CourseDataRow = CourseDataTable.Rows(Row_number);  
  
        CourseName.Text = CourseDataRow.Item(0);  
        Duration.Text = CourseDataRow.Item(1);  
        Fees.Text = CourseDataRow.Item(2);  
    }  
    Catch (Exception ex){  
        MessageBox.Show("Show Details Exception : " + ex.Message);  
    }  
}
```


}

Table Used: Course; **Database:** NorthWind


Table - dbo.course			
	Column Name	Data Type	Allow Nulls
	cour	varchar(20)	<input type="checkbox"/>
	duration	int	<input type="checkbox"/>
	fees	int	<input type="checkbox"/>

Fig 4.2-4: Course Table used in the above scenario



Context:

- Disconnected Architecture is prescribed in case of updation spanning multiple rows, as it reduces the network traffic.
- Constraints enforce restrictions on the data in a DataTable for maintaining data integrity.
- Relationships enable navigation from one table to another within a DataSet.



Practice Session:

- Mr. Hadlee heads Customer relations of Hadlee Inc. The work involved deals with viewing the customer details and tracking the orders placed by them through the CustomerID. Prepare an application regarding the same where the customer details table has a foreign key constraint with customer orders table. Customer Details table contains fields like CustomerID(pk), Customer Name, Customer Location and Customer Payments. Customer Orders table contains fields like OrderID(pk), CustomerID(fk), OrderPlacedDate, OrderDeliveryDate and PaymentSpecifications.



Check List:

- Importance of Disconnected Architecture.
- The components(objects) involved in DataSet Object Model
- Methods provided by the DataRow object.
- Types of constraints and their use.



Common Errors:

- Incorrect information provided in the connection string or query string.
- Two tables with the same name, irrespective of whether they are in the same or different namespace can be created with ADO.NET 2.0. However this is not possible in the older version, ADO.NET



Exceptions:

- InvalidOperationException might occur if the source table that is being accessed is invalid.

- Argument Exception might occur if the details provided in the connection string are insufficient.



Lessons Learnt:

- If a name is not specified for the DataSet, the name is set to "New Dataset".
- DataRelation objects can be used to programmatically fetch related child records from a parent record or a parent record from a child record.
- DataTable names must be unique, so that an Exception will be thrown when duplicate table names are used.



Best Practices:

- It is better to use a DataRow to retrieve the data from a table than a DataSet, as it helps to reduce the coding and resource requirement.
- Always declare the objects of DataRow or DataSet globally and fill the DataSet in the Form Load method to prevent redundant coding across the program.



Topic: Data Relationships

Estimated Time: 50 mins.



Objectives: At the end of the session, the participant will be able to understand:

- Data Relation concepts.
- Usage of DataRelation objects.
- Linking of two tables using DataRelation.



Presentation:

- Relationships are created between matching columns in the parent and child tables having an identical DataType.
- Relationships can also cascade various changes from the parent DataRow to its child rows.
- Creation of a DataRelation is preceded by verification if the relationship can be established.
- Following its addition to DataRelationCollection, the relationship is maintained by rejecting any changes to invalidate it.
- DataRelation objects can be accessed through:
 - Relations property of the DataSet.
 - ChildRelations and ParentRelations properties of the DataTable.



Scenario:

Mr.Chang Yu owns Xmart Marketing Company which deals with different categories of products. In order to promote them, the products have to be categorized. Build a windows-based application which displays the products belonging to each category in detail by implementing the DataRelation object.



Demonstration/Code Snippet:



Fig 4.5-1 : Output Showing the Products in different categories

Step 1: Open visual studio and create a new windows application

Step 2: Import the following namespaces

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

Step 3: Define Category_Products class

```
Public Class Category_Products  
{  
    SqlConnection objConn;  
    SqlDataAdapter adap;  
    DataSet ds;  
    DataRow dtrParentCategory;  
    DataRow dtrChildProduct;  
}
```

Step 4: This methods fills the DataSet with categories and products and finally adds a relation among them.

```
Private void view_products_Click(ByVal sender As System.Object, ByVal e As  
System.EventArgs) Handles view_products.Click  
  
    Try  
    {  
        ' Create a Connection to the Server  
        Create a Connection to the Server  
        objConn = New SqlConnection();  
        objConn.ConnectionString =  
            "server=hstslc011;uid=sa;password=satyam;database=northwind";  
        objConn.Open();  
        ' Create a new instance of DataSet  
        ds = New DataSet();  
  
        ' Fill the DataSet with the Categories table data  
        adap = New SqlDataAdapter("SELECT * FROM Categories", objConn);  
        adap.Fill(ds, "Categories");  
  
        ' Again fill the DataSet with the Products table data  
        adap.SelectCommand = New SqlCommand("SELECT * FROM Products", objConn);  
        da.Fill(ds, "Products");  
    }  
  
    Catch(SqlException exc)  
        MsgBox(exc.Message);  
    Finally  
        ' disposing the connection  
        objConn.Dispose();  
    ' To Create the Data Relationship  
  
    ds.Relations.Add("Cat_Prod", ds.Tables("Categories").Columns("CategoryID"  
    ), ds.Tables("Products").Columns("CategoryID"));  
  
    ' For each Parent loop (category) the child loop(products) is to print '
```

all the products in that category

```

For Each dtrParentCategory In ds.Tables("Categories").Rows
    lblDisplay.Text &= vbCrLf & "CategoryName -- " &
    dtrParent("CategoryName").ToString & vbCrLf & "ProductNames --";

    For Each dtrChildProduct In dtrParent.GetChildRows("Cat_Prod")
        lblDisplay.Text &= ", " & dtrChild("ProductName").ToString;
    Next

    LblDisplay.Text &= vbCrLf; ' vbCrLf is to print the text in newline
Next
}

```

Tables Used: Categories, Products; **Database:** NorthWind;

Table - dbo.Products			
Column Name	Data Type	Allow Nulls	
ProductID	int	<input type="checkbox"/>	
ProductName	nvarchar(40)	<input type="checkbox"/>	
SupplierID	int	<input checked="" type="checkbox"/>	
CategoryID	int	<input checked="" type="checkbox"/>	
QuantityPerUnit	nvarchar(20)	<input checked="" type="checkbox"/>	
UnitPrice	money	<input checked="" type="checkbox"/>	
UnitsInStock	smallint	<input checked="" type="checkbox"/>	
UnitsOnOrder	smallint	<input checked="" type="checkbox"/>	
ReorderLevel	smallint	<input checked="" type="checkbox"/>	
Discontinued	bit	<input type="checkbox"/>	

Figure 4.5-2: Products Table

Table - dbo.Categories			
Column Name	Data Type	Allow Nulls	
CategoryID	int	<input type="checkbox"/>	
CategoryName	nvarchar(15)	<input type="checkbox"/>	
Description	ntext	<input checked="" type="checkbox"/>	
Picture	image	<input checked="" type="checkbox"/>	

Figure 4.5-3: Categories table



Context:

- DataRelation object enhances performance as much less data is retrieved when compared to a server side join.
- DataRelation objects facilitate data consistency through automatic deletion and updation of child records as per the parent record.
- DataRelation objects facilitate navigation through automatic repositioning of child records as and when the parent changes.



Practice Session:

- Mercantile Retail Co. Ltd. is a leading shopping mart. Prepare an application that allows adding new brands and products to its list. The application should allow adding relations and constraints between the brands and their products. The fields involved in the categories table and products table are similar to the scenario.



Check List:

- Importance of Data Relationships
- Advantages of using DataRelation object.



Common Errors:

- Different DataType of the columns in the parent and child table on which the relation has been defined.
- Attempting to modify the values of the child column or parent column in a way that violates the unique constraint or foreign key constraint.



Exceptions:

- ArgumentException might occur if no relation exists at the specified index or if it is not a part of the collection or if it is already present while trying to redefine.
- DataException might occur if the child and parent tables belong to different DataSets.



Lessons Learnt:

- Relationships are created between matching columns in the parent and child tables having identical DataType.
- Relationships can cascade various changes from the parent DataRow to its child rows.



Best Practice:

- To control how values are changed in child rows, add a **ForeignKeyConstraint** to the **ConstraintCollection** of the **DataTable** object. The **ConstraintCollection** determines what action is to be taken when a value in a parent table is deleted or updated.



Topic: Modifying Data in a DataTable

Estimated Time: 50 mins.



Objectives: At the end of the session, the participant will know how to:

- Insert a new row in a DataTable
- Modify existing Rows
- Delete a row.



Presentation:

- Activities that can be performed on tables in the database are allowed in DataTables in a DataSet also.
- Data can be added, viewed, edited, and deleted in the DataTable.
- Errors and events can be monitored.
- Changes made to the DataTable can be accepted or rejected.

Inserting New Row

- To add a new row, declare a new variable as Typed DataRow.
- A new DataRow object is returned when the NewRow method of DataTable is called.

Modifying Row

- When changes are made to the column values in a DataRow, the changes are immediately placed in the Current state of the row.
- The RowState is then set to be Modified and the changes can be accepted or rejected using the AcceptChanges () or RejectChanges () methods of the DataRow.

Deleting Row

- There are two methods available:
 - The **Remove** method of the DataRowCollection object: It removes a DataRow from the DataRowCollection
 - The **Delete** method of the DataRow object: It only marks the row for deletion. The actual removal occurs when the application calls the AcceptChanges method.

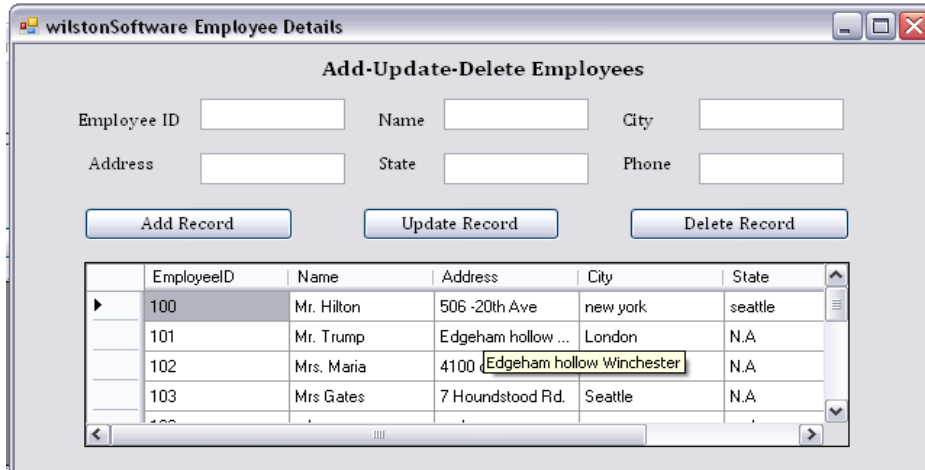


Scenario:

Mr. Whitherspoon works as a Database Administrator at Wilston Software Solutions. To maintain the details of Employees of the company, the options of add, update and delete are required. Create an application which allows the user to view, add, update, delete the records in the Employee_details table using DataSet and DataTable. The Employee details includes EmployeeID, Name, Address, City, State, Phone.



Demonstration/Code Snippet:



EmployeeID	Name	Address	City	State
100	Mr. Hilton	506 -20th Ave	new york	seattle
101	Mr. Trump	Edgeham hollow ...	London	N.A
102	Mrs. Maria	4100 (Edgeham hollow Winchester		N.A
103	Mrs Gates	7 Houndstood Rd.	Seattle	N.A

Figure 4.6-1: Application to Add-Update-Delete Employee Details

Step 1: Open visual studio and create a new windows application

Step 2: Import the following namespaces

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

Step 3: Define the EmployeeDetails class

```
Public Class EmployeeDetails
```

```
    SqlConnection DbCon = new SqlConnection();
    // Declaring new DataSet, Datatable, Adapter for EmployeeDetails
    DataSet EmployeeDataSet = new DataSet();
    SqlDataAdapter EmpAdapter;
    DataTable EmpDataTable;
```

Step 3: EmployeeDetails_Load Method fills the EmployeeDetails into the DataSet and shows the output in the EmployeesGridview

```
private void EmployeeDetails_Load(object sender, System.EventArgs e)
{
    DbCon.ConnectionString =
    "server=htslc011;database=northwind;uid=eltp;password=satyam";
    try {
        // Create a new Connection and SqlDataAdapter
        SqlCommand EmpDetailsCmd = new SqlCommand();
```



```

EmpDetailsCmd.CommandType = CommandType.Text;
EmpDetailsCmd.CommandText = "select * from employee_details";
EmpDetailsCmd.Connection = DbCon;
EmpAdapter = new SqlDataAdapter(EmpDetailsCmd);

// Create command builder. This line automatically generates the
update commands for you, so you don't have to provide or create your own.
SqlCommandBuilder myDataRowsCommandBuilder = new
SqlCommandBuilder(EmpAdapter);

// Set the MissingSchemaAction property to AddWithKey because Fill
will not cause primary key & unique key information to be retrieved unless
AddWithKey is specified.
EmpAdapter.MissingSchemaAction = MissingSchemaAction.AddWithKey;

// Fill the DataSet and assign it to the Employee dataTable
EmpAdapter.Fill(EmployeeDataSet, "Employee_Details");
EmpDataTable = EmployeeDataSet.Tables("Employee_Details");
// To represent the Employee Details in the grid view
EmployeesGridView.DataSource = EmpDataTable;
}
catch (Exception ex) {
    MessageBox.Show("Error " + ex.Message());
}
}

```

Step 4 : Add_Record Method adds a new record to datatable and updates to the database

```

private void Add_Record_Click(object sender, System.EventArgs e)
{
    // Create a new row using dataTable object by calling NewRow Method
    DataRow dr = EmpDataTable.NewRow;
    // Assign the values to the datarow object columnwise
    dr("Name") = Emp_Name.Text;
    dr("Address") = Emp_Address.Text;
    dr("City") = Emp_City.Text;
    dr("State") = Emp_State.Text;
    dr("Phone") = Emp_Phone_no.Text;

    // Add the row to the datatable
    EmpDataTable.Rows.Add(dr);

    // Update the DataSet to the database using DataAdapter by calling update
method.
    EmpAdapter.Update(EmployeeDataSet, "Employee_Details");
    MessageBox.Show("New Record Added");
    // fill the DataSet to get the latest Updates
    EmpAdapter.Fill(EmployeeDataSet, "Employee_Details");
}

```

Step 5 : Update_Record Method updates an existing record in the datatable and finally updates it to database

```
private void Update_Record_Click(object sender, System.EventArgs e)
{
    try {
        // Find the specific row using find Method and assigning the row to
        the datarow object
        DataRow dr = EmpDataTable.Rows.Find(Emp_ID.Text);

        if (dr == null) {
            Interaction.MsgBox("No Record with Id " + Emp_ID.Text);
        }
        else {
            // Call for beginEdit Method, do all updations to the specific row
            and finally Endedit Method is called
            dr.BeginEdit();
            dr("Name") = Emp_Name.Text;
            dr("Address") = Emp_Address.Text;
            dr("City") = Emp_City.Text;
            dr("State") = Emp_State.Text;
            dr("Phone") = Emp_Phone_no.Text;
            dr.EndEdit();
            // Update the DataSet to the database using DataAdapter by calling
            update method and fill the DataSet to get the latest Updates
            EmpAdapter.Update(EmployeeDataSet, "Employee_Details");
            EmpAdapter.Fill(EmployeeDataSet, "Employee_Details");
            MessageBox.Show("Record Updated with ID :" + Emp_ID.Text);
        }
    }
    catch (FormatException ex) {
        MessageBox.Show("Invalid ID");
    }
}
```

Step 6 : Delete_Record Method deletes an existing record in the datatable and finally updates it in the database

```
private void Delete_Record_Click(object sender, System.EventArgs e)
{
    try {
        // Find the specific row using find Method and assigning the row to
        the datarow object
        DataRow dr = EmpDataTable.Rows.Find(Emp_ID.Text);
        if (dr == null) {
            Interaction.MsgBox("No Record with Id " + Emp_ID.Text);
        }
        else {
            // Call for delete method in order to delete the row
            dr.Delete();
        }
    }
}
```

```
// Update the DataSet to the database using DataAdapter by calling
update method and fill the DataSet to get the latest Updates
EmpAdapter.Update(EmployeeDataSet, "Employee_Details");
EmpAdapter.Fill(EmployeeDataSet, "Employee_Details");
MessageBox.Show("Record has been deleted with ID " + Emp_ID.Text);
}
}
catch (FormatException ex) {
    MessageBox.Show("Invalid ID");
}
}
```

Table Used: Employee_Details; **Database:** NorthWind

Table - dbo.employee_details			Table - dbo.employee_deta
Column Name	Data Type	Allow Nulls	
EmployeeID	int	<input type="checkbox"/>	
Name	varchar(30)	<input checked="" type="checkbox"/>	
Address	varchar(30)	<input checked="" type="checkbox"/>	
City	varchar(30)	<input checked="" type="checkbox"/>	
State	varchar(30)	<input checked="" type="checkbox"/>	
Phone	varchar(15)	<input checked="" type="checkbox"/>	

Figure 4.6-2: Employee_details table used in scenario



Context:

- Manipulation of data is done in a DataTable whenever changes are to be updated to the database using a Disconnected Architecture.



Practice Session:

- Hernst financial services deals with providing solutions to their client's stock market queries and providing investment suggestions. Prepare an application that will allow the clients to register themselves by filling in personal details. The application should allow Hernst services to view, modify, add and delete the details of their clients. The clients table consists of fields like Client ID, Client Name, Client Address, E-mail ID and Payment Mode.



Check List:

- Steps involved in adding a new row.
- How to Position a new Row.
- Use of AcceptChanges() and RejectChanges() methods in modifying a row.
- Deletion of a row.



Common Errors:

- Attempting to edit a row which is read-only or such that it violates the constraints set in it.

- Generating a command using **CommandBuilder** and trying to work on data that does not exist.



Exceptions:

- **RowNotInTable** Exception occurs while attempting to manipulate **DataRow**s that are not in the **DataTable**.
- **DeletedRowInaccessible** Exception might occur while trying to edit deleted row.
- **NoNullValue** Exception might occur while attempting to edit the column in a row whose **DBNull** value is set to false, to a null value.



Lessons Learnt:

- **MissingSchemaAction** property determines the action to be taken when existing **DataSet** schema does not match incoming data.
- Creating an instance of the **DataTable** is not necessary to make changes to the table in the database. It can also be done by simply declaring the **DataTable** object.



Best Practices:

- Use of the **CommandBuilder** should be limited to design time or ad-hoc scenarios.
- When the command property is null (by default null), the **CommandBuilder** generates a command for a **DataAdapter** command property.
- The **CommandBuilder** does not overwrite the explicitly set value of command property.
- Set the command property to null to generate a command by **CommandBuilder** for a command property.



Topic: Sorting and Filtering

Estimated Time: 50 mins.



Objectives: At the end of the session, the participant will be able to know:

- To sort and filter data by using the Select method.
- Describe DataView object.
- Create a DataView object.
- Sort and filter the rows in DataTable by using a DataView object



Presentation:

Select Method

- Select method of a DataSet object gets an array of DataRow objects.
- It creates a set of pointers to rows within the original DataSet.

DataView object

- It is an object that presents a subset of data from a DataTable.
- A DataView object acts similarly to a layer on top of the DataTable, providing a filtered and sorted view of the table contents.

How to Define a DataView

- DataView object can be created in two ways:
 - By using Visual Studio .NET development environment's graphical tools
 - Programmatically

Working in a DataTable

- Finding a Specific Row: Applying the **FindBy method** on the table helps to find a row in the DataTable via the **primary key**.
- Selecting Multiple Rows in a DataTable: Query the table similar to how you would query it in a database with the **Select Method**.
- Performing Aggregate Calculations: This can be done through the methods available in DataTable object like the **DataTable. Compute** method.



Scenario:(DataView)

Mr. Hoops, the owner of Nightly Surveys wants to build an application wherein the customers can know the number of employees pertaining to a particular location for the listed companies. The user can view the data sorted by name or number of employees through this application.



Demonstration/Code Snippet:



Figure 4.7-1: Output Screenshot showing CompanyDetails

Step 1: Open visual studio and create a new windows application

Step 2: Import the following namespaces

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

Step 3: Define the Sort_filter class

```
Public Class Sort_filter
{
    SqlConnection DbCon = new SqlConnection();
    // Create a new DataSet object and dataview object
    DataView CompanyDataView;
    DataSet CompanyDataSet = new DataSet();
    SqlDataAdapter adap = new SqlDataAdapter();
}
```

Step 4: The following Form Load method loads the DataSet with company details and displays it in the DataView

```
private void Sort_filter_Load(object sender, System.EventArgs e)
{
    // Defining the Connection String
    DbCon.ConnectionString =
    "server=hstslc011;database=northwind;uid=eltp;password=satyam";
    try {
        // Create a new Connection and SqlDataAdapter
```

```

        // Create a new Command to get all the details of the company
        SqlCommand CompanyDetails = new SqlCommand();
        CompanyDetails.CommandType = CommandType.Text;
        CompanyDetails.CommandText = "select name,location,No_Of_Employees
from Companies";
        EmpDetailsCmd.Connection = DbCon;
        // Define new DataAdapter and fill the DataSet using DataAdapter with
        CompanyDetails
        da.SelectCommand = CompanyDetails;
        da.Fill(CompanyDataSet, "Companies");
        // To create new instance of dataview from the DataSet
        CompanyDataView = new DataView(CompanyDataSet.Tables("Companies"));

        // Set rowFilter property that filters the rows based on the condition
        CompanyDataView.RowFilter = "Location='Bangalore'";
        // Sort property to sort the order of the dataview
        CompanyDataView.Sort = "name asc";
        CompanyDataView.RowStateFilter = DataViewRowState.CurrentRows;

        // To represent the data present in dataview in the gridview
        CompanyGridView.DataSource = CompanyDataView;
    }

    catch (Exception ex) {
        MessageBox.Show(ex.Message);
    }
}

```

Step 5: This method filters rows based on the location by setting the rowFilter property of the dataview

```

private void Sel_Location_SelectedIndexChanged(object sender, System.EventArgs e)
{
    CompanyDataView.RowFilter = "Location='" + Sel_Location.SelectedItem.ToString
+ "'";

```

```

        CompanyDataView.RowStateFilter = DataViewRowState.CurrentRows;
        CompanyGridView.DataSource = CompanyDataView;
    }

```

Step 6: This method sorts the rows based on the column by setting the sort property of the dataview

```

private void Sort_by_SelectedIndexChanged(object sender, System.EventArgs e)
{
    CompanyDataView.Sort = Sort_by.SelectedItem.ToString + " Asc";
    CompanyDataView.RowStateFilter = DataViewRowState.CurrentRows;
    CompanyGridView.DataSource = dv;
}

```

Table Used: companies; **Database:** Northwind

Table - dbo.companies			
	Column Name	Data Type	Allow Nulls
	id	int	<input type="checkbox"/>
	name	nvarchar(45)	<input checked="" type="checkbox"/>
	location	nvarchar(45)	<input checked="" type="checkbox"/>
	No_of_employees	int	<input checked="" type="checkbox"/>

Figure 4.7-2: Table used in the scenario.



Scenario:(DataTable)

Mr. Hamilton, the managing director of Aldos Systems wants to view the record containing the details of the employees. Employee table contains the employee's name, his department code, Grade, Age, date of joining, gender, salary and marital status.

Create an application fulfilling the following requirements using DataTable.

- View records of the employees using the id's.
- View the list of employees
- No. of employees in each department.



Demonstration/Code Snippet:



The screenshot shows a window titled "Aldos Systems". At the top, there are two input fields: "Emp Id" with the value "1" and "Department Id" which is empty. Below these are two buttons: "Get Employee Details" and "View Employees". The main area of the window displays the details for Employee ID 1 in a two-column layout:

Emp ID	1	DOJ	1/1/1995
Name	David BLain	Gender	M
Dept Code	GEN	Salary	16000
Age	35	Marital Status	Y

At the bottom of the details section, there are two buttons: "Previous" and "Next".

Figure4.7-3: Showing the EmployeeDetails for a given EmployeeID



The screenshot shows a Windows application window titled "Aldos Systems". Inside, there is a "Statistics" dialog box with the text "Employees in this Department - 5" and "Avg Salary - 5860", and an "OK" button. To the right, there is a "Department Id" field with the value "soft" and a "View Employees" button. Below these, there is a form with the following fields: "Emp ID" (2), "Name" (sadasd2), "Dept Code" (SOFT), "Age" (27), "DOJ" (3/1/1995), "Gender" (M), "Salary" (6000), and "Marital Status" (N). At the bottom of the form are "Previous" and "Next" buttons.

Figure4.7-4: Showing the Employees in the Department and general Statistics, Next-Previous helps to navigate among records

Step 1: Open visual studio and create a new windows application.

Step 2: Import the following namespaces.

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

Step 3: Define the EmployeeDept class.

```
Public Class EmployeeDept

    SqlDataAdapter da = new SqlDataAdapter();
    DataSet EmpDataSet = new DataSet();
    DataTable EmpDataTable;
    int i;
    //integer 'i' represents the record number
    SqlConnection DbCon = new SqlConnection();
    DataRow[] EmployeesArray;
```

Step 4: The following load method fills the DataSet with EmployeeDetails and finally loading the dataTable with DataSet.

```
private void EmployeeDept_Load(object sender, System.EventArgs e)
{
    DbCon.ConnectionString =
    "server=hstslc011;database=northwind;uid=eltp;password=satyam";
    try {
        //Create a new Command to get all the details of the Employee
```

```

SqlCommand EmployeeCommand = new SqlCommand();
EmployeeCommand.CommandType = CommandType.Text;
EmployeeCommand.CommandText = "select * from Employee";
EmployeeCommand.Connection = DbCon;
//Defining new DataAdapter and filling the DataSet using DataAdapter
with Employee Details

    da.SelectCommand = CompanyDetails;
    da.MissingSchemaAction = MissingSchemaAction.AddWithKey;
    da.Fill(EmpDataSet, "employee");
    //Loading the dataTable with DataSet containing EmployeeDetails
    //To create a datatable from DataSet
    EmpDataTable = EmpDataSet.Tables("Employee");
}

catch (Exception ex) {
    MessageBox.Show("Exception : " + ex.Message);
}

}

```

Step 5: The following GetEmployeeDetails button click method gets the EmployeeDetails for a given EmployeeID.

```

private void GetEmployeeDetails_Click(object sender, System.EventArgs e)
{
    try {
        // Find () method helps to find a row in the DataTable via the primary
key.
        DataRow EmpDetailsDataRow = EmpDataTable.Rows.Find(Emp_Id.Text);
        if (EmpDetailsDataRow == null) {
            Interaction.MsgBox("No Record with ID " + Emp_Id.Text);
        }
        else {
            // Following method (defined Later) is used to show the details of
the record (row) that has been passed as argument

            Show_record(EmpDetailsDataRow);
        }
    }
    catch (FormatException ex) {
        MessageBox.Show("Enter a valid ID");
    }
}

```

Step 6: The following ViewEmployees button click method gets the Employees present in a given department.

```

private void ViewEmployees_Dept_Click(object sender, System.EventArgs e)
{

```

```

' Perform Aggregate Calculations through the methods available in
DataTable object like DataTable.Compute method. It is a must to pass two
arguments to this method:
    •Expression to compute
    •Filter
    If there is no filter then string.empty has to be specified
instead.
object No_Of_Records = (int)EmpDataTable.Compute("Count(emp_code)",
"dept_code='" + Dep_Code.Text + "'");
object avg_sal = (int)EmpDataTable.Compute("avg(salary)", "dept_code='"
+ Dep_Code.Text + "'");
' Select Multiple Rows in a DataTable using select method which returns
an array of DataRow objects based on the condition and the sort order
EmployeesArray = EmpDataTable.Select("dept_code='" + Dep_Code.Text +
"'", "emp_Code asc");
' here integer 'i' represents the record number to be shown
i = 0
' passing the 'i'th record from the Arrayof DataRows
Show_record(EmployeesArray.GetValue(i));

MessageBox.Show("Employees in this Department - " + No_Of_Records +
Strings.Chr(13) + "Avg Salary - " + avg_sal, "Statistics");

}

```

Step 7: The following method is used to get the next EmployeeRecord in the Department on the Next button click event.

```

private void Next_Record_Click(object sender, System.EventArgs e)
{
    if (i < EmployeesArray.Length - 1) {
        // here integer 'i' is incremented to point to the next record
        i = i + 1;
        Show_record(EmployeesArray.GetValue(i));
    }
}

```

Step 8: The following method is used to get the Previous EmployeeRecord in the Department on Previous button click event.

```

private void Previous_Record_Click(object sender, System.EventArgs e)
{
    if (i > 0) {
        // here integer 'i' is decremented to point to the previous record
        i = i - 1;
        Show_record(EmployeesArray.GetValue(i));
    }
}

```

Step 9: This method is used to show all the elements in the row.

```
public void Show_record(DataRow dr)
{
    Emp_Code.Text = dr("Emp_Code");
    Emp_Name.Text = dr("Emp_Name");
    Dept_Code.Text = dr("Dept_Code");
    Age.Text = dr("Age");
    Date_Join.Text = dr("date_join");
    Gender.Text = dr("Gender");
    salary.Text = dr("salary");
    married.Text = dr("married");
}
```

Table Used: employee; **Database:** NorthWind

Table - dbo.employee		
Column Name	Data Type	Allow Nulls
emp_code	int	<input type="checkbox"/>
emp_name	varchar(25)	<input type="checkbox"/>
dept_code	varchar(4)	<input checked="" type="checkbox"/>
grade	varchar(2)	<input checked="" type="checkbox"/>
age	int	<input checked="" type="checkbox"/>
date_join	datetime	<input checked="" type="checkbox"/>
gender	varchar(1)	<input checked="" type="checkbox"/>
salary	int	<input checked="" type="checkbox"/>
married	varchar(1)	<input checked="" type="checkbox"/>

Figure 4.7-5: Table used in the above scenario



Context:

- DataView object allows data binding on both Windows Forms and Web Forms.
- Controls can be bound to DataView which is useful for storing default criteria of sorting and filtering.



Practice Session:

- Time Certifications is an online agency that provides certificate level courses of .NET. Aspirants are required to take an online examination for getting certified. Prepare an application that lists the results of the examination in a descending order of marks. It also displays the number of examinees, the average score and groups them according to the grades obtained. Exam table contains Exam_Code and Exam_Name whereas Students table contains StudentID, Student Name, Exam_Code, Marks_Obtained and Grade_Obtained.



Check List:

- Use of Select method to filter data.
- Creating a DataView object.

- Methods to work on a table.
- Method used to perform aggregate calculations using a DataTable



Common Errors:

- Generated results might not be in accordance with the expected results during sorting and filtering operations if the relevant expressions do not take care of DataTable's CaseSensitive property.
- Insufficient information supplied in the Connection String.



Exceptions:

- DeletedRowInaccessible Exception might be raised while trying to access a deleted row.
- When the cast is not performed with CONVERT method, then InvalidCastException might occur.
- Argument Exception might occur when the expression in the DataTable's Select method does not comply with the required rules.



Lessons Learnt:

- The primary key (Emp_id) of the row to be found is required in the Fill method of the DataTable object.
- Value returned from the Compute method of the DataTable is object type and must be type cast into integer.
- In case of a missing filter in the Compute method, string.empty has to be added as the argument.
- Sort property of the DataView object get or set the sort column and sort order for the DataView object.

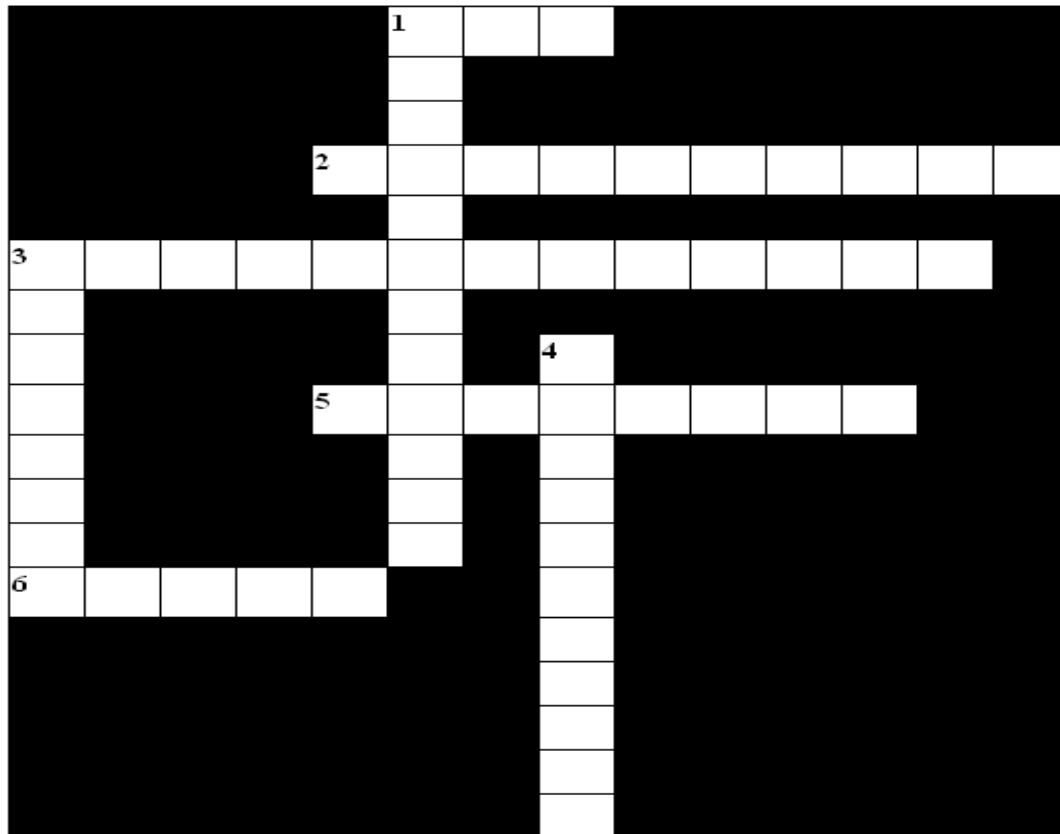


Best Practices:

- When creating a **DataView** object, use the **DataView** constructor that takes the **Sort**, **RowFilter**, and **RowStateFilter** values as constructor arguments (along with the underlying **DataTable**). The result in the index is built once. Create an "empty" DataView and set the Sort, RowFilter or RowStateFilter properties after the results in the index have been built at least twice.

Crossword: Unit-4

Estimated Time: 10 mins.



Across:






1. Which method is used to add a new row in DataTable (3).
2. Which method is used to check that some changes have been made to DataSet since it was loaded (10).
3. To revert or abandon all changes since the DataSet was loaded, which method is used. (13).
5. Which class represents a complete table or can be small section of rows, depending on some criteria (8).
6. What is the nature of a DataSet (5).

Down:

1. Which method is used to commit all the changes since last time Accept Changes has been executed (12).
3. Which method is used to remove a "Data Row" object from "Data Table" depending on index position of the "Data Table"(8).
4. The ability of ADO.NET components to link the data to the objects is known as (11).

5.0 Reading and Writing XML with ADO.NET

Topics

-  5.1 Creating XSD schemas
-  5.2 Strongly typed DataSets
-  5.3 Loading schemas and data into DataSets
-  5.4 Writing XML from a DataSet
-  5.5 Crossword





Topic: XSD Schemas

Estimated Time: 40 mins.



Objectives: At the end of the session, the participant will be able to:

- Understand an XSD schema.
- Create and save an XSD schema.



Presentation:

What Is an XSD Schema?

- XSD schema describes the structure of an XML document, as well as the constraints on data.
- XSD schema can contain the following information:
 - Representation of relationships between data items.
 - Representation of constraints.
 - Specification of the data types of each individual element and attribute in an XML document that complies with the XSD schema.
- Reasons to use XSD schemas:
 - To import data and know the structure of the data.
 - To describe the structure of data being exported to another consumer



Scenario:

Mr. Smith of JPM Company wants to create an XSD schema for each. Link simple types with complex types for consistency of the data. The schema includes creating simple, complex and attribute groups and incorporating simple into complex types.



Demonstration/Code Snippet:

Step 1: Open Visual Studio.net and open a new windows application project. Let's call it XMLEmployees

Step 2: In the solution explorer add a new item - XML Schema. Name this Schema EmployeeListSchema.xsd

Step 3: From the toolbox drag a complex type onto the design surface. Rename the complextype1 to Employer.

Step 4: In the row below this, add 2 elements, namely employer_id, which is of type long and employer_name, which is a string.



CT	Employer	
E	employer_id	long
E	employer_na	string

Step 5: Drag an attribute onto the design surface. Name this attributes Country and make sure it is a string.

A	Country	string
---	---------	--------

CT	Employer	
E	employer_id	long
E	employer_na	string

Step 6: Drag and drop the attribute over the Employer complex type underneath the 2 elements.

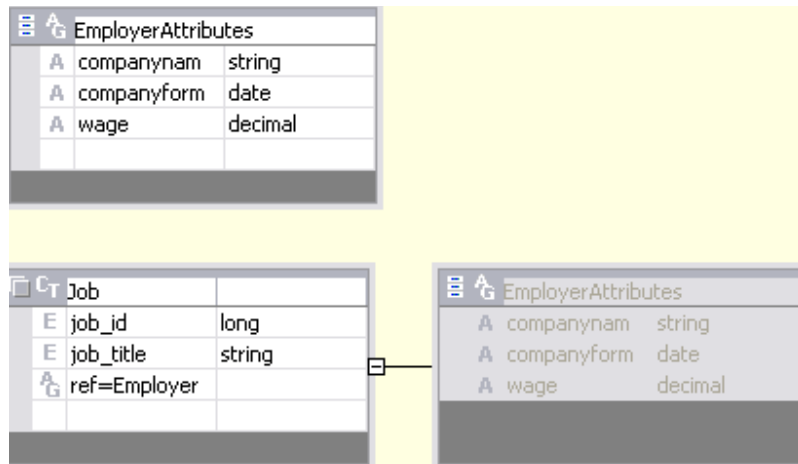
CT	Employer	
E	employer_id	long
E	employer_na	string
A	Country	string

Step 7: Drag and drop an attribute group onto the design surface from the toolbox - call it Employer Attributes.

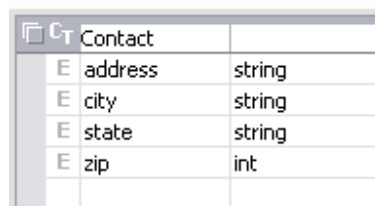
Step 8: Add some attributes to the attribute group such as Company Name of type string, company Formed Date of type Date, and wages of type decimal.

AG	EmployerAttributes	
A	companyname	string
A	companyform	date
A	wage	decimal

Step 9: Add a complex type from the toolbox and call it Job. Add some elements to this; add a job_id element of type long, employment_id element of type long, jobtitle element of type string. Lastly add an attribute group. This will create a link to a blank attribute group - if it is given the same name as the previous attribute group it will create a copy of the previous attribute group (Employer Attributes). This will add a reference=attribute group to this attribute group from the complex type.

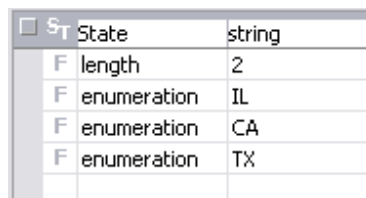


Step 11: Again add a complex type and call it Contact. Add more elements like address of type string, city of type string, state of type string, and lastly zip of type int.

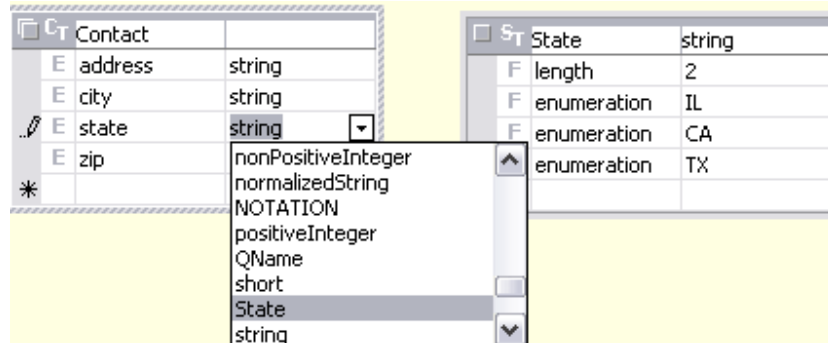


Step 12: Drag a simpleType onto the design surface and name it as State of type string. Add some properties to the simpleType by selecting the row below and from the drop down choose length and enter a value of 2 across the length property.

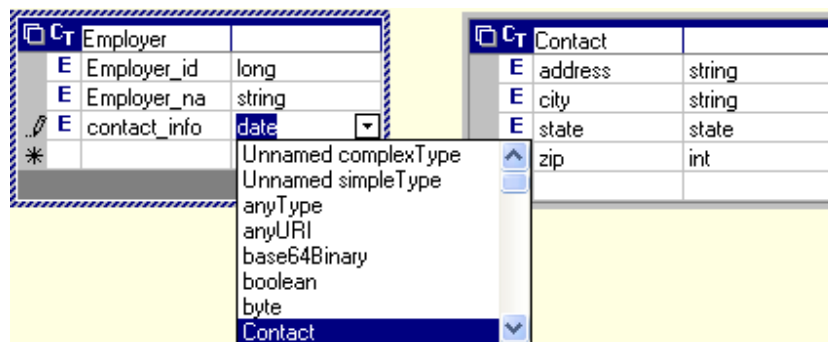
Step 13: Drag a facet object from the toolbox into the above SimpleType, make the property as enumeration and the value as IL (for the state). Create some more facets, say 3(IL, CA and TX perhaps).



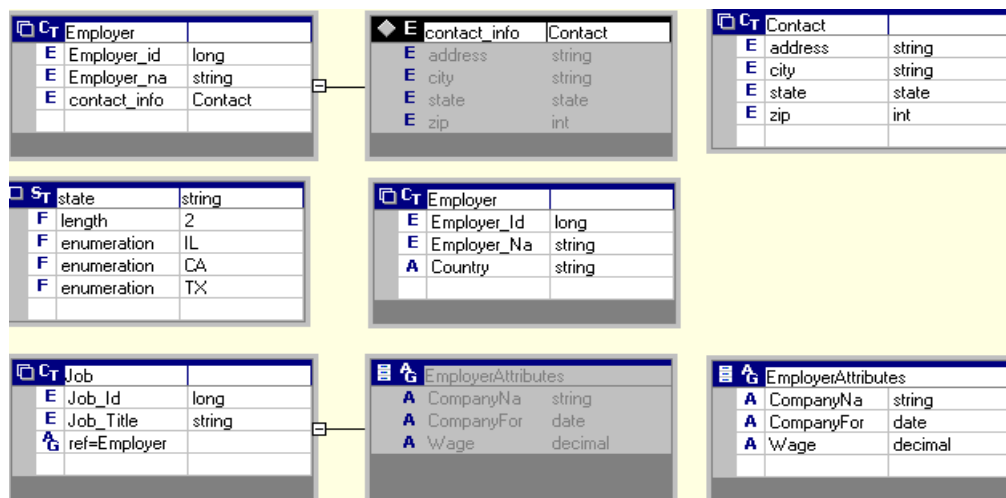
Step 14: Go back to the complex type for contact and select the type as state (referring to above simple type) for the element state.



Step 15: Add a complex type called Employer and add elements such as Employer_id of type long, Employer_name of type string, and contact_info but this time of type contact. This will add a link in the designer to our Contact complex type.



Step 17: Lastly look at the XML view in the designer or at the generated code for seeing the XML which has been built up from this exercise. Code behind Design



XML code for the above Schema:

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema id="XMLSchema1" targetNamespace="http://tempuri.org/XMLSchema1.xsd"
elementFormDefault="qualified" xmlns="http://tempuri.org/XMLSchema1.xsd"
xmlns:mstns="http://tempuri.org/XMLSchema1.xsd"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:complexType name="Employer">
    <xs:sequence>
      <xs:element name="Employer_Id" type="xs:long" />
      <xs:element name="Employer_Name" type="xs:string" />
    </xs:sequence>
    <xs:attribute name="Country" type="xs:string" />
  </xs:complexType>

  <xs:attributeGroup name="EmployerAttributes">
    <xs:attribute name="CompanyName" type="xs:string" />
    <xs:attribute name="CompanyFormedDate" type="xs:date" />
    <xs:attribute name="Wage" type="xs:decimal" />
  </xs:attributeGroup>

  <xs:complexType name="Job">
    <xs:sequence>
      <xs:element name="Job_Id" type="xs:long" />
      <xs:element name="Job_Title" type="xs:string" />
    </xs:sequence>
    <xs:attributeGroup ref="EmployerAttributes" />
  </xs:complexType>

  <xs:complexType name="Contact">
    <xs:sequence>
      <xs:element name="address" type="xs:string" />
      <xs:element name="city" type="xs:string" />
      <xs:element name="state" type="state" />
      <xs:element name="zip" type="xs:int" />
    </xs:sequence>
  </xs:complexType>

  <xs:simpleType name="state">
    <xs:restriction base="xs:string">
      <xs:length value="2" />
      <xs:enumeration value="IL" />
      <xs:enumeration value="CA" />
      <xs:enumeration value="TX" />
    </xs:restriction>
  </xs:simpleType>

  <xs:complexType name="Employer">
    <xs:sequence>
      <xs:element name="Employer_id" type="xs:long" />
      <xs:element name="Employer_name" type="xs:string" />
      <xs:element name="contact_info" type="Contact" />
    </xs:sequence>
```

```
</xs:complexType>  
</xs:schema>
```



Context:

- XSD schema defines the structure of the data that is to be read into a DataSet.
- XSD defines the structure of the data being sent from the DataSet to an XML file.



Practice Session:

- Mr. Schmidt is the Database Administrator of Roark Solutions Inc. He wants to prepare an XSD Schema for the company's employees. The schema should generate separate tables for Technical and Human Resource departments but, having the same structure which contains fields like Employee ID, Employee Name, Employee Address, E-mail and Joining Date.



Check List:

- Creation of the XSD Schema.
- Mapping XSD Schema Information to Relational Structure.



Common Error:

- Deleting any complex type such as Contact (address, City, state, Country) in the corresponding data type column before removing the references used.



Exception:

- The code developed might not work with some old applications which are not in compliance with the W3C standards (existence of such applications is rare).



Lessons Learnt:

- Schema is a collection of <element> tags, and <simpleType> and <complexType> elements that aggregate other elements into a cohesive block.
- Complex type is a type definition for elements that may contain attributes and elements.
- Constraints and information about the values of attributes or elements with text-only content is determined by SimpleType.
- Attribute type provides additional information about ComplexType.
- Attribute group can be defined only as a child of the schema element.
- AttributeGroup can be included in any other Attribute Group or ComplexType using 'ref' attribute.



Best Practices:

- Define an item as a ComplexType if it contains sub elements in it.
- To reuse an item's content in other items; define it as a complex type.
- Given a set of attributes for an element, it's better to define an Attribute Group and include the group using 'ref' attribute.
- Use an attribute if the information is required to be in a standard DTD-like definition such as ID, IDREF, or ENTITY.



Topic: Strongly Typed DataSets

Estimated Time: 40 mins.



Objectives: At the end of the session, the participant will be able to:

- Define an XML Schema
- Use user-defined types



Presentation:

- XML Schema definition provides the syntax for representing elements and attributes in a XML document.
- Instance document is an **XML** document conforming to an **XML Schema** type.

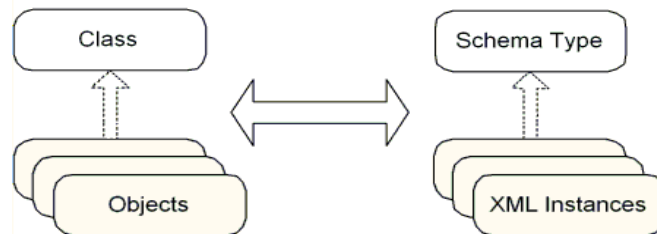


Figure 5.2-1: Comparison of XML Schema with object-oriented concepts.

- An XSD Schema element can be of:
 - **Simple Type:** Users can define their own custom simple types, whose value spaces are subsets of the predefined built-in types.
 - **Complex Type:** Different simple types (or value spaces) can also be arranged into a structure known as a complex type.



Scenario:

Stanley Mobile Services deals with providing support to some of the leading mobile companies. It decides to fix a schema for registering its user's personal details. Prepare the required XML schema.



Demonstration/Code Snippet:

Step 1: Create a simple type in XSD schema named Custid

```
<xsd:simpleType name="custid">  
<xsd:restriction base="xsd:string">
```

'Restrict the length of DataType to 11

```
<xsd:length value="11">
```

'It should be displayed in the specific format like "123-44-5555"

```
<xsd: pattern value="\d{3}\-\d{2}\-\d{4}" />
</xsd:restriction>
</xsd:simpleType>
```

Step 2: Create a simple type in XSD schema named State containing restricted values only.

```
<xs:simpleType name="State">
<xs:restriction base="xs:string">

'Restrict the length of DataType to 2
<xs:length value="2" />

'It should contain only three names "IL, CA, and TX"

<xs:enumeration value="IL" />
<xs:enumeration value="CA" />
<xs:enumeration value="TX" />
</xsd:restriction>
</xs:simpleType>
```

Step 3: Create a complex type in XSD schema named Customer element

```
<xsd:complexType name="Customer">
<xsd:sequence>
'It should contain child elements Custid, Name, Date of Birth, Address and
State.
<xsd:element name="Custid" type="xsd:string">
<xsd:element name="Name" type="xsd:string">
<xsd:element name="DateOfBirth" type="xsd:date">
<xsd:element name="Address" type="xsd:string"/>
<xsd:element name="State" type="xsd:string"/>
</xsd:sequence>
</xsd:complexType>
```

Step 4: Now incorporate simple types Custid, State into complex type Customer.

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<xsd:element name="Customer"
<xsd:complexType>
<xsd:sequence>
'Here user defined simple DataType custid is incorporated into custid of
complex type
<xsd:element name="Custid" type="xsd:custid">
<xsd:element name="Name" type="xsd:string">
<xsd:element name="DateOfBirth" type="xsd:date">
<xsd:element name="Address" type="xsd:string"/>
'Here user defined simple DataType state is incorporated into state of complex
type
<xsd:element name="State" type="xsd:state"/>
</xsd:sequence>
</xsd:complexType>
```




Context:

- XSD schema is used to define own data type from the existing data type.
- It provides the opportunity for re-usability
- XML Schema avoids the need to learn any new syntax.



Practice Session:

- Explore the various annotations that can be used with typed DataSets.
- Prepare XML schemas for generating a strongly Typed DataSet containing the Clients table containing ClientID(pk) and Client Name fields, and Products table containing ProductID(pk), ClientID(fk), Product Name and Product Details fields, for Stanley Mobile Services. Prepare an application that adds new records to this DataSet without violating the associated foreign key constraint.



Check List:

- Basics of XML Schema definitions including simple and complex type definitions.



Common Errors:

- When XSD is added to XMLSchemaCache Object, the error shown is“Schema Is Non-Deterministic”.
- Specifying the type as SimpleType instead of ComplexType when defining a DataSet.
- Incorrect nesting and closing of XML tags.



Exception:

- When nesting is undesired, msData:Relationship annotation can be used to specify parent-child relationships between elements that are not nested.



Lessons Learnt:

- XML Schema provides an expressive type system for XML capable of defining predefined and new Types.
- SimpleType definitions allow you to define custom value spaces for text-only elements and attributes.
- ComplexType definitions allow you to arrange Simple Types into structures.
- ComplexType child element of a schema element generates a table in the DataSet



Best Practice:

- Create new data types with comprehensible naming and optimized length.



Topic: Loading Schemas and Data into DataSets

Estimated Time: 40 mins.



Objectives: At the end of the session, the participant will be able to:

- Load DataSet Schema Information from XML



Presentation:

- The schema of a DataSet can be defined programmatically:
 - Created by the Fill or Fill Schema methods of a DataAdapter
 - Loaded from an XML document.
- To load DataSet schema information from an XML document, use either:
 - ReadXmlSchema
 - InferXmlSchema



Scenario:

Mr. Brook of JPMorgan company wants to load all the employees' details, which is in the XML document, into database with the help of DataSets.



Demonstration/Code Snippet:

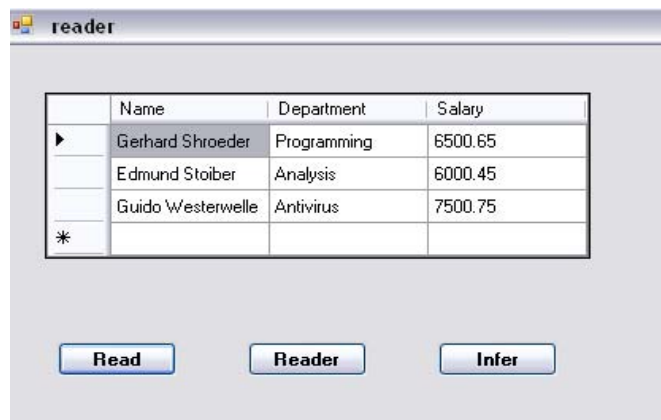


Figure 5.3-1:Output screenshot showing employee details

Step 1: Open the Visual Studio and create a windows application.

Step 2: The following code appears on the screen
using System.Data;

```
PublicPartial class xml_read : Form {
```

```
private DataSet ds;  
public xml_read()  
{  
    InitializeComponent();  
}  
}
```

Step 2: Write the following code in the Read button click function. An XML Schema file name is passed as parameter to the ReadXml method. The file retrieved from the XML file is displayed in a GridView through a DataSet

```
private void ReadXML_Click(object sender, EventArgs e)  
{  
  
    ds = new DataSet();  
    ds.ReadXml("");  
    DataGridView1.DataSource = ds.Tables[0];  
  
}
```

Step 3: The following code is entered in Reader button click subroutine. An IOstreamreader object is passed as parameter to the ReadXmlSchema method. It reads the schema of the XML file whose path is provided and displays it in the GridView.

```
Private void Read_Schema_Click(object sender, EventArgs e)  
{  
  
    DataSet thisDataSet = new DataSet();  
    System.IO.StreamReader myStreamReader = new  
System.IO.StreamReader("C:\Documents and  
Settings\slclab\Desktop\customer3.xml");  
    thisDataSet.ReadXmlSchema(myStreamReader);  
    myStreamReader.Close();  
    DataGridView1.DataSource = thisDataSet.Tables[0];  
  
}
```

Step 4: An XML Schema file name is passed as parameter to the InferXmlSchema method.

```
private void infer_Schema_Click(object sender, EventArgs e)  
{  
  
    string ns;  
    ns = "urn:schemas-microsoft-com:officedata";  
    ds = new DataSet();  
    ds.InferXmlSchema("C:\\Documents  
andSettings\\slclab\\Desktop\\customer3.xml", new string[] {ns});  
}
```

```
DataGridView1.DataSource = ds.Tables[0];  
}
```



Context:

- XSD schema is loaded into DataSet in order to create a relational data structure in the DataSet



Practice Session:

- Glitz Telecommunications has recently taken over Mason Telecommunications. The two companies maintain different servers and had been using different technologies. The management decides to share the information through XML file transfer before getting the technologies normalized. Prepare an application that reads the information about the employees from the XML File and loads it into the Employees table of the database. Employees table contains fields like Employee_ID, Employee_Name, Contact_Info, and Dept_ID.



Check List:

- Load an XSD schema into a DataSet.
- Examine metadata
- Load data in XML format into a DataSet



Common Errors:

- Incorrect file path provided for StreamReader.
- Not closing the StreamReader



Exceptions:

- Security Exception is raised when FileIO permission is not set to read.
- ArgumentNullException is raised when a new instance for a DataSet is not created before reading XML into it.
- Argument Exception might occur if the values specified in any argument are invalid.



Lessons Learnt:

- ReadXmlSchema allows loading or inferring of DataSet schema information from the document containing (XSD) schema, or an XML document with in line XML Schema.
- InferXmlSchema allows inferring the schema from the XML document while ignoring certain specified XML namespaces.



Best Practices:

- When loading a DataSet from an XML file, you can load the schema of the DataSet from XSD Schema, or you can predefine the tables and columns before loading the data.
- If no XSD Schema is available and it is not known which tables and columns to define for the contents of an XML file, the schema can be inferred based on the structure of the XML document.



Topic: Writing XML from a DataSet

Estimated Time: 40 mins.



Objectives: At the end of the activity, the participant will be able to:

- Write XML data from a DataSet
- Create an XML representation of a DataSet, in order to transport the DataSet across HTTP



Presentation:

- XML representation of a DataSet can be written with or without its schema.
- XSD is used to write schema information which is included inline with the XML.
- Schema contains the DataSet table definitions as well as the relation and constraint definitions.
- XML representation can be written to a file, a stream, an XmlWriter, or a string.
- Two ways of presenting XML representation of a DataSet:
 - GetXml () returns the XML representation of the DataSet as string, without schema information.

```
Dim xmlDS As String = custDS.GetXml ()
```

 GetXmlSchema () writes the schema information from the DataSet to a string.
 - WriteXml () writes a DataSet to a file, stream or XmlWriter.



Scenario:

Mr. Brook of Hampshire University wants to load the details of all his students from a database to a XML document with the help of DataSets.



Demonstration/Code Snippet:

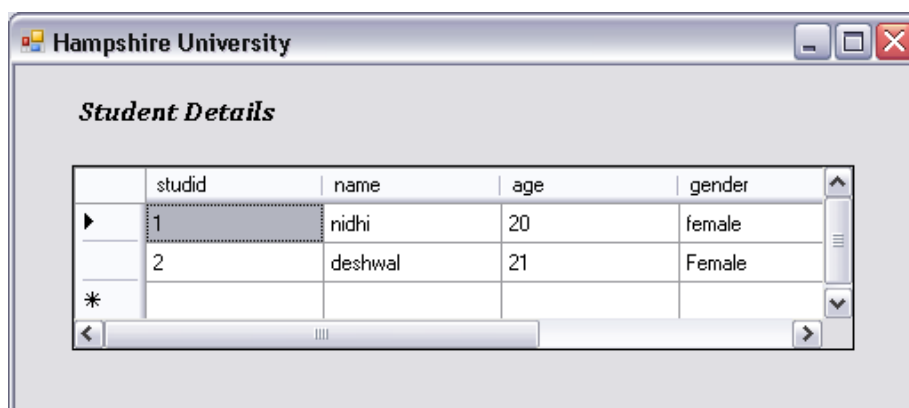


Figure 5.4-1: Output screen shot showing student details in DataSet

Step1: Create a new windows application in Visual Studio.

Step 2: Import the following namespaces

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

Step 3: Defining the Sort_filter class

```
public class Student_Detail  
{  
    'defining variables  
    private SqlConnection DbCon;  
    private SqlCommand StudDetailscmd;  
    private DataSet StudDataSet = new DataSet();  
    private SqlDataAdapter StudAdap;
```

Step 4: Student_Details_Load Method fills a DataSet with student table which is to be written to an XML file with ignore schema option

```
private void Student_Details_Load(object sender, System.EventArgs e)  
{  
    DbCon = new SqlConnection();  
    ' defining the coonection string and command  
    DbCon.ConnectionString = "server=hstslc011;database=Northwind;uid=eltp  
;pwd=satyam";  
    StudDetailscmd = new SqlCommand("select * from univstudent", DbCon);  
    ' Assigning command to the data adapter and filling the studentDataSet  
    StudAdap = new SqlDataAdapter(StudDetailscmd);  
    StudAdap.Fill(StudDataSet, "univstudent");  
    StudentsGridView.DataSource = StudDataSet.Tables[0];  
    ' Using writeXml method the following DataSet is written to the xml file  
    StudDataSet.WriteXml("student.xml", XmlWriteMode.IgnoreSchema);  
}
```

XML Output

```
<?xml version="1.0" standalone="yes"?>  
<NewDataSet>  
    <univstudent>  
        <studid>1</studid>  
        <name>nidhi</name>  
        <age>20</age>  
        <gender>female</gender>  
        <email>nidhi@stayam.com</email>  
        <course>RDMS</course>  
    </univstudent>  
    <univstudent>  
        <studid>2</studid>  
        <name>deshwal</name>  
        <age>21</age>  
        <gender>Female</gender>  
        <email>nidi@satyam.com</email>
```

```
<course>HTML</course>
</univstudent>
</NewDataSet>
```

Step5 : Filling a DataSet with student table which is to be written to an XML file with writeschema option

```
StudDataSet.WriteXml ("student.xml", XmlWriteMode.WriteSchema);
```

XML Output

```
<?xml version="1.0" standalone="yes"?>
<NewDataSet>
  <xs:schema id="NewDataSet" xmlns=""
    xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:msdata="urn:schemas-
    microsoft-com:xml-msdata">
    <xs:element name="NewDataSet" msdata:IsDataSet="true"
      msdata:UseCurrentLocale="true">
      <xs:complexType>
        <xs:choice minOccurs="0" maxOccurs="unbounded">
          <xs:element name="univstudent">
            <xs:complexType>
              <xs:sequence>
                <xs:element name="studid" type="xs:int" minOccurs="0" />
                <xs:element name="name" type="xs:string" minOccurs="0" />
                <xs:element name="age" type="xs:int" minOccurs="0" />
                <xs:element name="gender" type="xs:string" minOccurs="0" />
                <xs:element name="email" type="xs:string" minOccurs="0" />
                <xs:element name="course" type="xs:string" minOccurs="0" />
              </xs:sequence>
            </xs:complexType>
          </xs:element>
        </xs:choice>
      </xs:complexType>
    </xs:element>
  </xs:schema>
  <univstudent>
    <studid>1</studid>
    <name>nidhi</name>
    <age>20</age>
    <gender>female</gender>
    <email>nidhi@stayam.com</email>
    <course>RDMS</course>
  </univstudent>
  <univstudent>
    <studid>2</studid>
    <name>deshwal</name>
    <age>21</age>
    <gender>Female</gender>
    <email>nidi@satyam.com</email>
    <course>HTML</course>
  </univstudent>
</NewDataSet>
```


Step 6: Filling a DataSet with student table which is to be written to an XML file with diffgram option

```
StudDataSet.WriteXml ("student.xml", XmlWriteMode.DiffGram);
```

XML Output

```
<?xml version="1.0" standalone="yes"?>
<diffgr:diffgram xmlns:msdata="urn:schemas-microsoft-com:xml-msdata"
xmlns:diffgr="urn:schemas-microsoft-com:xml-diffgram-v1">
  <NewDataSet>
    <univstudent diffgr:id="univstudent1" msdata:rowOrder="0">
      <studid>1</studid>
      <name>nidhi</name>
      <age>20</age>
      <gender>female</gender>
      <email>nidhi@stayam.com</email>
      <course>RDMS</course>
    </univstudent>
    <univstudent diffgr:id="univstudent2" msdata:rowOrder="1">
      <studid>2</studid>
      <name>deshwal</name>
      <age>21</age>
      <gender>Female</gender>
      <email>nidi@satyam.com</email>
      <course>HTML</course>
    </univstudent>
  </NewDataSet>
</diffgr:diffgram>
```



Context:

- XML writing methods help in re-creating the structure of existing DataSets to newly typed DataSets.
- XML writing methods increases the inter-operability and re-usability of the applications using DataSet.



Practice Session:

- GE is a world leader in the appliances market. It has decided to add dishwashers to its appliances list. Prepare an application that reads the schema of an existing product category from an XML File which contains fields like Category ID, Category Name, and Category Details, and writes it to a DataSet containing the categories table. Also write back the schema and the data to an XML document.



Check List:

- Writing a schema to a file, reader or stream.
- Writing information in a DataSet object to a file or stream.
- Writing changes to a DataSet.



Common Errors:

- Incorrect file path provided for StreamReader.
- Not closing the StreamReader



Exceptions:

- Security Exception is raised when FileIO permission is not set to write.
- ArgumentNullException is raised when a new instance for a DataSet is not created before reading XML into it.
- Argument Exception might occur if the values specified in any argument are invalid.



Lessons Learnt:

- Destination of XML output is the first parameter which is passed to WriteXml. For example, pass a string containing a file name, a System.IO.TextWriter object, and so on
- An optional second parameter can be passed to XmlWriteMode to specify how the XML output is to be written.
- Columns of a table can also be mapped to XML Elements, Attributes and Text.

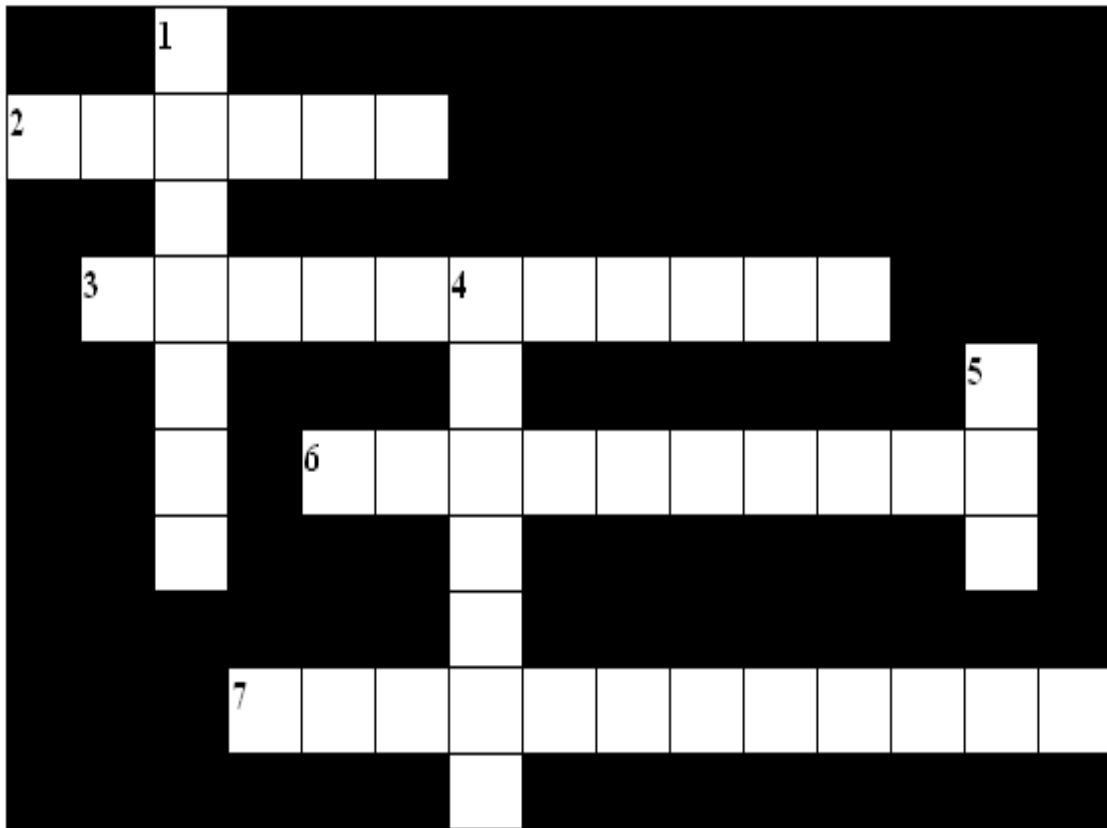


Best Practices:

- Use the DiffGram mode of WriteXmlSchema method while writing a DataSet as XML data if both current and original values of rows are required by the application.

Crossword: Unit-5

Estimated Time: 10 mins.



Across:









2. The type of cursor that prevents users from seeing records that others have added and prevents access to deleted records (6).
3. This acts as the bridge between the database and the disconnected objects in the ADO.NET object model(11)
6. The method that can be used to extract only the modified rows from a DataSet (10).
7. DataSet that is derived from the base DataSet class (12)

Down:

1. The type of cursor that enables the viewing of additions, changes, and deletions made by other users and allows forward and backward movement through a recordset object(7).
4. The object that is central to supporting disconnected, distributed data scenarios using ADO.NET (6).
5. The schema provides the syntax and defines a way in which elements and attributes can be represented in an XML document(3)

6.0 Building DataSets from Existing Data Sources

Topics

-  6.1 Configuring a Data Adapter to Retrieve Information
-  6.2 Populating a DataSet using a Data Adapter
-  6.3 Configuring a Data Adapter to update the underlying Data Source
-  6.4 Persisting changes to a Data Source
-  6.5 How to handle Conflicts
-  6.6 Typed and Un-Typed DataSets
-  6.7 Binding to DataGrid control
-  6.8 Crossword





Topic: DataAdapter

Estimated Time: 50mins.



Objectives: At the end of the session, the participant will be able to:

- Define a DataAdapter
- Use the properties of a DataAdapter object.
- Implement the methods available in the DataAdapter object.



Presentation:

Defining DataAdapter

- DataAdapter is the bridge between the database and the disconnected objects in the ADO.NET object model.
- DataAdapter class represents a set of database commands and connection used for filling a DataSet and updating the data source.
- DataAdapter object submits pending changes in the DataSet back to the database.

DataAdapter Properties

Properties are instances of the SqlCommand or OleDbCommand class:

- **SelectCommand:** Retrieves rows from the data source.
- **InsertCommand:** Writes inserted rows from the DataSet into the data source.
- **UpdateCommand:** Writes modified rows from the DataSet into the data source.
- **DeleteCommand:** Deletes rows in the data source.

DataAdapter Methods

- **Fill:** Adds or refreshes rows from a data source and places them in a DataSet table by using the SelectCommand property.
- **Update:** Changes the data in the data source to match the data in the DataSet by calling the Insert, Update or Delete commands over the DataSet.



Scenario:

Mr.Neil, Managing director of Rosewood company, uses an application which lets him see all the salesman residing in different cities, insert new salespersons' data into the database, update the existing data and delete the names of the persons who have left the company.



Demonstration/Code Snippet:



	name	age	city	country
▶	mali	20	Mumbai	India
*	sachin	26	Mumbai	India

Figure 6.1-1: Output screenshot showing the selected rows whose city="Mumbai"

Step 1: Open Visual Studio and create a windows application.

Step 2: Create a windows form as shown above. Include the following namespace.

```
using System.Data.SqlClient;

public class RosewoodCompany : Form
{
    private SqlConnection con;
    private SqlCommand emp_cmd;
    private SqlDataAdapter emp_ad;
    private DataSet emp_ds;

    public RosewoodCompany()
    {
        base.New();
        InitializeComponent;
    }
}
```

Step 3: Create a Connection to the Server

```
private void Da_Property_Load(object sender, EventArgs e)
{
    Private str="server = hstslc011;database = Northwind;uid = eltp;
    pwd = satyam;";
    con = new SqlConnection();
    con.ConnectionString = str;
}
```

Step 4: Enter the following code in **Select button** click. The function below initializes the SelectCommand property of DataAdapter.

```
private void btnselect_Click(object sender, EventArgs e)
{
    SqlParameter pcountry;
    SqlParameter pcity;
    string
cmd_str = "SELECT * FROM slcstudent WHERE City= @City AND Country = @Country";
    SqlCommand emp_cmd = new SqlCommand(cmd_str, con);

    `Parameters are passed to it
    pcity = new SqlParameter("@City", city_txt.Text);
    pcity.SqlDbType = SqlDbType.VarChar;
    pcountry = new SqlParameter("@Country", country_txt.Text);
    pcountry.SqlDbType = SqlDbType.VarChar;
    emp_cmd.Parameters.Add(pcity);
    emp_cmd.Parameters.Add(pcountry);

    `Create a DataAdapter object and assign the command object to its
    Selectcommand property.
    emp_ad = new SqlDataAdapter();
    emp_ad.SelectCommand = emp_cmd;

    `Create DataSet object and fill it using the DataAdapter. Bind the
    DataSet to the gridview for display.
    emp_ds = new DataSet();
    emp_da.Fill(emp_ds, "slcstudent");
    sales_GridView1.DataSource = emp_ds.Tables[0];
}
```

Step 5: Write the following code in the **Insert button** click event. Use Insertcommand property of DataAdapter to add new data.

```
private void btninsert_Click(object sender, EventArgs e)
{
    SqlParameter pcountry;
    SqlParameter pname;
    SqlParameter page;
    SqlParameter pcity;
    string cmd_str = "INSERT INTO slcstudent VALUES(@Name,@Age,@City;@Coun
try)";
    emp_cmd = new SqlCommand(cmd_str, con);
    `Parameters are passed to it
```

```

pname = new SqlParameter("@Name", name_txt.Text);
pname.SqlDbType = SqlDbType.VarChar;
page = new SqlParameter("@Age", Convert.ToInt32(age_txt.Text));
page.SqlDbType = SqlDbType.Int;
pcity = new SqlParameter("@City", city_txt.Text);
pcity.SqlDbType = SqlDbType.VarChar;
pcountry = new SqlParameter("@Country", country_txt.Text);
pcountry.SqlDbType = SqlDbType.VarChar;
emp_cmd.Parameters.Add(pname);
emp_cmd.Parameters.Add(page);
emp_cmd.Parameters.Add(pcity);
emp_cmd.Parameters.Add(pcountry);
`Create a DataAdapter object and assign the command object to its
Insertcommand property.
emp_ad = new SqlDataAdapter();
emp_ad.InsertCommand = emp_cmd;
emp_update(emp_ds);
MessageBox.Show("1 row inserted");
}

```

Step 6: Write the following code in the **Update button** click event. Use Updatecommand property of DataAdapter to modify existing data.

```

Private void btnupdate_Click(object sender, EventArgs e)
{
    SqlParameter pname;
    SqlParameter pcity;
    SqlParameter pcountry;
    string cmd_str = "UPDATEslcstudent SET city=@City,                country=@Country W
    HERE name = @Name";
    emp_cmd = new SqlCommand(cmd_str, con);
    pname = new SqlParameter("@Name", name_txt.Text);
    pname.SqlDbType = SqlDbType.VarChar;
    pcity = new SqlParameter("@City", city_txt.Text);
    pcity.SqlDbType = SqlDbType.VarChar;
    pcountry = new SqlParameter("@Country", country_txt.Text);
    pcountry.SqlDbType = SqlDbType.VarChar;
    emp_cmd.Parameters.Add(pname);
    emp_cmd.Parameters.Add(pcity);
    emp_cmd.Parameters.Add(pcountry);

    `Create a DataAdapter object and assign the command object to its
    Updatecommand property.
    emp_ad = new SqlDataAdapter();
    emp_ad.UpdateCommand = emp_cmd;
    emp_ad.update(emp_ds);
    MessageBox.Show("1 row updated");
}

```



```
}
```

Step 7: Write the following code in the **Delete button** click event. Use Deletecommand property of DataAdapter to delete existing data.

```
Private void btndelete_Click(object sender, EventArgs e)

{

SqlParameter pname;
string cmd_str = "DELETE FROM slcstudent WHERE name = @Name";
emp_cmd = new SqlCommand(cmd_str, con);
pname = new SqlParameter("@Name", name_txt.Text);
pname.SqlDbType = SqlDbType.VarChar;
emp_cmd.Parameters.Add(pname);

'Create a DataAdapter object and assign the command object to its
Deletecommand property.

emp_ad = new SqlDataAdapter();
emp_ad.DeleteCommand = emp_cmd;
emp_ad.update(emp_ds);
MessageBox.Show("1 row deleted");

}
```

Table Used: (slcstudent; database: northwind)

Table - dbo.slcstudent		hstslc011.Nort...SQLQuery1.sql*
Column Name	Data Type	Allow Nulls
▶ name	varchar(30)	<input type="checkbox"/>
age	int	<input type="checkbox"/>
city	varchar(30)	<input type="checkbox"/>
country	varchar(30)	<input type="checkbox"/>
		<input type="checkbox"/>

Figure 6.1-2: Table used in the scenario



Context:

- DataAdapter object retrieves and saves data between a DataSet and a data source.
- DataAdapter class helps to manage disconnected functionality.



Practice Session:

- Mr. Jackie Yang heads the Market research division of JAGUAR. Prepare an application that allows him to add, modify and delete the results of his research using the DataAdapter class. The research involves gathering the customer feedback, regional acceptance, sales figures and

comparison with rivals. This information is to be inputted into the MarketResearch table for each ProductID.



Check List:

- Working with ADO.NET's disconnected environment.
- Importance of DataAdapter object
- Displaying, adding and modifying data using DataAdapter.



Common Errors:

- Incorrect information provided in the connection string or query string.
- Row added containing data that violates a constraint required on a DataColumn in the DataSet.
- Data added to DataSet is incompatible with common language runtime.



Exceptions:

- InvalidOperationException Exception might occur while using Update method if the source table is invalid.
- DBConcurrency Exception might occur if an INSERT, UPDATE or DELETE statement results in zero records affected.



Lessons Learnt:

- Creating a DataAdapter programmatically.
- Methods and Properties involved in DataAdapter.



Best Practices:

- Supposing a search function on your corporate Web site needs to return a list of matches on a Web page. It would be inappropriate to use a DataAdapter and DataSet in this situation because the results will be thrown away as soon as the page is created. There is no reason to cache this data in a DataSet.
- To perform multiple modifications on an SQL table, you take a copy of a subset of the table and store that copy in a middle tier or a user tier as a DataSet.



Topic: Populating a DataSet Using a DataAdapter Estimated Time: 30 mins.



Objectives: At the end of the activity, the participant will be able to:

- Infer additional constraints for a DataSet
- Populate a DataSet from multiple DataAdapters.



Presentation:

Infer Additional Constraints for a DataSet

- To create and modify DataSet schema at run time:
 - Set the **MissingSchemaAction** property on the DataAdapter determining how the schema is created and the action to be taken while retrieving DataTables or DataColumnns that are not present in the DataSet schema.
The following code retrieves the primary key and the foreign key information:
`da.MissingSchemaAction = MissingSchemaAction.AddWithKey`
 - **FillSchema** method builds a new schema, applies any existing table mappings to the retrieved schema (using the SelectCommand object), and configures the DataSet with the transformed schema.
`adapter.FillSchema(DataSet, SchemaType.Mapped);`

Populate a DataSet from multiple DataAdapters

- Each DataAdapter fills a separate table in the DataSet.
- Referential integrity between related tables in the database can be preserved by controlling the order in which updates are written back to the database.

Configuring a DataAdapter to Update the Underlying Data Source

- Each DataRow has a RowState property which indicates whether the row has been modified, inserted or deleted from the DataSet since the DataSet was first populated.
- The DataSet maintains two sets of data for each row:
 - **DataRowVersion.Current**: Current version of data in the DataRow.
 - **DataRowVersion.Original**: Original version of data in the DataRow.




Scenario1 (Multiple DataAdapters):

Selective Banking Services is a banking firm which wants to view the details of the accounts present in the bank and the corresponding customer details. Create a simple windows application which fulfills this requirement. Use the concept of filling a DataSet with multiple DataAdapters.



Demonstration/Code Snippet:



	Act_Id	Branch	City	State
▶	1	Somajiguda	Hyderabad	AP
*				

	Name	Age	Email	Address
▶	Swaroop	22	kavali@satya	101,NGO co

Fill

Figure 6.2-1: Output snapshot showing the two tables from same DataSet

Step 1: Create a windows application in Visual Studio.

Step 2: Import the following namespace
`using System.Data.SqlClient;`

Step 3: Declaring the following variables

```
public class MultDataAdapter : Form
{
    private SqlConnection Con;
    private DataSet CustDataSet;
    public MultDataAdapter()
    {
        base.New();
        InitializeComponent();
    }
}
```

Step 4: Enter the following code in the MultDataAdapter load method. It opens the connection to the database.

```
private void MultDataAdapter_Load(object sender, EventArgs e)
{
    str = "server=hstslc011;database=Northwind;uid=eltp;pwd=satyam";
    Con = New SqlConnection(str);
    Con.Open();
}
```

```
CustDataSet = New DataSet;  
  
}
```

Step 5: The following code is added in the Fill button click method.

```
private void Fillbtn_Click(object sender, EventArgs e)  
{  
  
    `create two command objects retrieving two tables  
    custstr = "select * from Acctdetails";  
    CustCmd = new SqlCommand(custstr, Con);  
    actstr = "select * from Custdetails";  
    ActCmd = new SqlCommand(actstr, Con);  
  
    `first adapter containing the data from custdetails table.  
    CustDa = new SqlDataAdapter(CustCmd);  
    CustDa.Fill(CustDataSet, "Custdetails");  
  
    `second adapter containing data from the acctdetails table.  
    AcctDa = new SqlDataAdapter(ActCmd);  
    AcctDa.Fill(CustDataSet, "Acctdetails");  
    MessageBox.Show("tables are filled");  
  
    `displays the two tables from the DataSet in two different DataGrids.  
    DataGrid1.DataSource = CustDataSet.Tables[0];  
    DataGrid2.DataSource = CustDataSet.Tables[1];  
}  
  
}
```



Scenario2 (Updating underlying data source):

The following example iterates through the rows in a DataSet table, and displays the RowState property for each row. If the RowState is **DataRowState.Added** or **DataRowState.Unchanged**, the current version of the row data is displayed. If the RowState is **DataRowState.Deleted**, the original version of the row data is displayed. If the RowState is **DataRowState.Modified**, the original and current versions of the row data are displayed to show how they differ.



Demonstration/Code Snippet:

```
` Declare a datarow object.  
DataRow row;  
  
` The following loop runs for each row in the DataSet.  
foreach (row in this.dsCustomers.Customers.Rows) {
```

```

string msg;

` If a new row is added or an existing row remains unchanged, then Current data
in the DataSet is displayed.
if (((row.RowState == DataRowState.Added)
    || (row.RowState == DataRowState.Unchanged)))
{
    msg = ("Current data:" + ("\r\n"
        + (row("CompanyName", DataRowVersion.Current) + (", " + row("C
ontactName", DataRowVersion.Current)))));
}
` Else if a row is deleted the Original version is displayed.
else if ((row.RowState == DataRowState.Deleted))
{
    msg = ("Original data:" + ("\r\n"
        + (row("CompanyName", DataRowVersion.Original) + (", " + row("
ContactName", DataRowVersion.Original)))));
}
` Else if an existing row has been modified, original version and the latest
version are displayed.
else if ((row.RowState == DataRowState.Modified))
{
    msg = ("Original data:" + ("\r\n"
        + (row("CompanyName", DataRowVersion.Original) + (", "
        + (row("ContactName", DataRowVersion.Original) + "\r\n"))));
    msg = (msg + ("Current data:"
        + (row("CompanyName", DataRowVersion.Current) + (", " + row("C
ontactName", DataRowVersion.Current)))));
}
MessageBox.Show(msg, ("RowState: " + row.RowState.ToString()));
}

```



Context:

- Filling a DataSet with multiple DataAdapter helps preserve referential integrity between related tables in the database.



Practice Session:

- E-Shoppe, an online retail shop, needs to periodically download data from its remote server. The structure of the data pertaining to products might change over time. Prepare an application that takes care of these changes with the help of various DataRow properties so that the user is never presented with out of date or outdated details of a product.
- A salesperson needs to retrieve customer information and information about orders placed by each customer, from the central database. To meet this requirement, you create a disconnected application that contains two DataAdapters: one to retrieve customer records, and the other to retrieve order records.



Check List:

- Filling the DataSet with the DataAdapter.
- Runtime modification of DataSet.
- Configuring a DataAdapter to Update the Underlying Data Source
- Persisting Changes to a Data Source



Common Errors:

- If the EnforceConstraints property is not set to false, no constraint can be enabled at the end of merge and all rows that are invalid show an error message.



Exceptions:

- During a merge, constraints are disabled. If constraints cannot be enabled at the end of merge, a ConstraintException is generated and the merged data is retained while the constraints are disabled.
- DeletedRowInaccessibleException Exception will be raised when the deleted row is accessed in the normal way. It can be done through the indexer on a DataRow



Lessons Learnt:

- The return value of the GetChanges() method is a filtered copy of the DataSet that can have actions performed on it and subsequently be merged back in using **Merge**.
- GetChanges method produces a second **DataSet** object that contains only the changes introduced into the original.
- When you call **AcceptChanges** on the **DataSet**, **DataRow** objects still in edit-mode end their edits successfully.
- **Merge** merges a specified DataSet and its schema with the current DataSet, preserving or discarding changes in the current DataSet.



Best Practices:

- To improve performance, set the EnforceConstraints property of DataSet to false before you fill the DataSet. This disables constraint checking while the data is loaded.
aDataSet.EnforceConstraints = False
- Generating a DataSet schema at run time is less efficient than building a Typed DataSet class at design time.
- Avoid the MissingSchemaAction property and the FillSchema method as they are slow, because they build the DataSet schema at run time.
- Calling the **AcceptChanges** of the **DataSet** enables you to invoke the method on all subordinate objects (for example: tables and rows) with one call.



Topic: Persisting Changes to a Data Source

Estimated Time: 40 mins.



Objectives: At the end of the session, the participant will be able to:

- Use the **GetChanges** method of a DataSet object
- Use the **Update** method of a DataAdapter object
- Use the **Merge** method to combine changes into the DataSet.
- Use the **AcceptChanges** method.



Presentation:

- Invoke the **GetChanges** method to create a second DataSet that features only the changes to the data.
- Invoke the **Merge** method to merge the changes from the second DataSet into the first DataSet.
- Call the **Update** method of the SqlDataAdapter (or OleDbDataAdapter) and pass the merged DataSet as an argument.
- Invoke the **AcceptChanges** method on the DataSet to persist changes. Alternatively, invoke the **RejectChanges** method to cancel the changes.

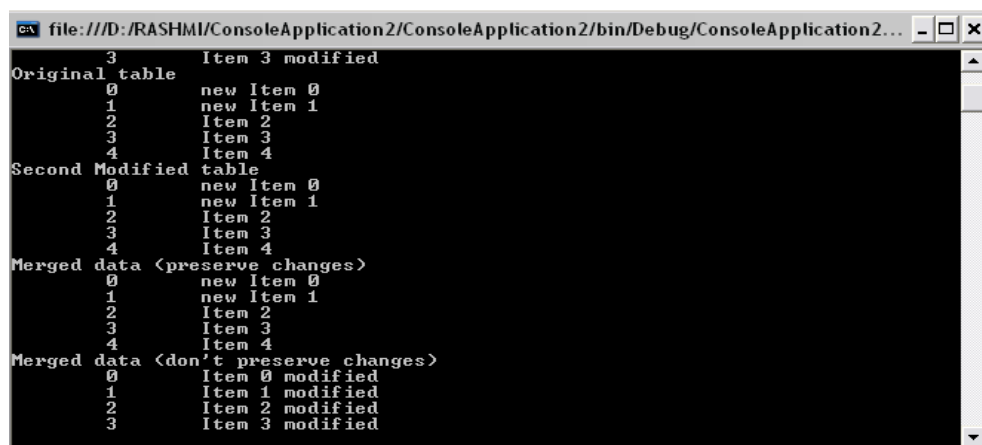


Scenario:

The following example creates a DataTable and a copy of it. It shows how the Merge () and AcceptChanges () methods work on a DataTable. If the changes are preserved, the current version of the row data is displayed. Orelse, the original version of data is retained.



Demonstration/Code Snippet:



```
file:///D:/RASHMI/ConsoleApplication2/ConsoleApplication2/bin/Debug/ConsoleApplication2...
3      Item 3 modified
Original table
0      new Item 0
1      new Item 1
2      Item 2
3      Item 3
4      Item 4
Second Modified table
0      new Item 0
1      new Item 1
2      Item 2
3      Item 3
4      Item 4
Merged data <preserve changes>
0      new Item 0
1      new Item 1
2      Item 2
3      Item 3
4      Item 4
Merged data <don't preserve changes>
0      Item 0 modified
1      Item 1 modified
2      Item 2 modified
3      Item 3 modified
```

Figure 6.4-1: Output screenshot showing preserved data after merge

Step 1: Open Visual Studio and create a console application.

Step 2: Include the following namespace.

```
using System.Data;

namespace Itemtable {

class Program
{
    private static void Main(string[] args)
    {
        DemonstrateMergeTable();
    }
}
}
```

Step 3: Creating a new datatable named `Itemtable` and adding columns to it.

```
private static void DemonstrateMergeTable()
{
    ' Create a new DataTable.
    DataTable Itemtable = new DataTable("Items");
    ' Add two columns to the table:
    DataColumn column = new DataColumn("id", typeof(int));
    column.AutoIncrement = true;
    Itemtable.Columns.Add(column);
    column = new DataColumn("item", typeof(string));
    Itemtable.Columns.Add(column);
    ' Set primary key column.
    Itemtable.PrimaryKey = new DataColumn[] {
        table1.Columns[0]};
    ' Add some rows.
    DataRow row;
    for (int i = 0; (i <= 3); i++) {
        row = Itemtable.NewRow();
        row["item"] = ("Item " + i);
        Itemtable.Rows.Add(row);
    }
    ' Accept changes.
    Itemtable.AcceptChanges();
    PrintValues(Itemtable, "Original values Itemtable");
    ' Using the same schema as the original table,
    ' modify the data for later merge.

    'Creating a copy of itemtable and name it as modifiedTable then make
changes in it
    DataTable modifiedTable = Itemtable.Copy();
    foreach (DataRow rowModified in modifiedTable.Rows) {
        rowModified["item"] = (rowModified["item"].ToString() + " modified
");
    }
}
```

```

    }
    modifiedTable.AcceptChanges();
    ' Change row values, and add a new row:
    Itemtable.Rows[0]["item"] = "new Item 0";
    Itemtable.Rows[1]["item"] = "new Item 1";
    row = Itemtable.NewRow();
    row["id"] = 4;
    row["item"] = "Item 4";
    Itemtable.Rows.Add(row);
    'Creating another copy of Itemtable named tableCopy
    ' Get a copy of the modified data:
    DataTable tableCopy = Itemtable.Copy();
    PrintValues(modifiedTable, "Modified table");
    PrintValues(Itemtable, "Original table");
    PrintValues(tableCopy, "Second Modified table ");
    ' Merge new data into the modified data.
    Itemtable.Merge(modifiedTable, true);
    PrintValues(Itemtable, "Merged data (preserve changes) ");
    tableCopy.Merge(modifiedTable, true);
    PrintValues(modifiedTable, "Merged data (don't preserve changes)");
    Console.ReadLine();
}

```

Step 6: Creating PrintValues method in order to print all the rows in datatable

```

private static void PrintValues(DataTable table, string label)
{
    ' Display the values in the supplied DataTable:
    Console.WriteLine(label);
    foreach (DataRow row in table.Rows) {
        foreach (DataColumn column in table.Columns) {
            Console.Write((" + ('\\t' + \"{0}\")), row[column, DataRowVersion
n.Current]);
        }
        Console.WriteLine();
    }
}
}
}

```



Context:

- When changes are made concurrently on the database in order to maintain consistency, GetChanges method is useful.
- Merge method allows to suggest changes in other applications in concurrent working environment.
- AcceptChanges gives the ability to incorporate or discard the changes in a DataSet which helps maintaining data consistency.



Practice Session:

- E-Rent is an online rental agency, a sister concern of Villa & Bros. Finances. Mr. Barrett is responsible for charting the agency's terms and conditions regarding renting. Officials of Villa & Bros. may suggest changes to him by manipulating the table. Mr. Barrett is responsible for evaluating the changes, making any additions and updating them back to the database. The fields of the RentPlan table are PlanID, PlanName, PlanT&C, PlanDuration and PlanDetails. Prepare a windows application for the above requirement.



Check List:

- GetChanges method of a DataSet object.
- Use the Update method of a DataAdapter object.
- Use the Merge method to merge changes into the DataSet.
- Use the AcceptChanges method.



Exception:

- InvalidOperationException might occur if the Merge method deals with an invalid source table.



Lessons Learnt:

- The return value of the GetChanges() method is a filtered copy of the DataSet that can have actions performed on it, and subsequently be merged back in using **Merge**.
- GetChanges method produces a second **DataSet** object that contains only the changes introduced into the original.
- When you call **AcceptChanges** on the **DataSet**, **DataRow** objects that are still in edit-mode end their edits successfully.
- **Merge** merges a specified DataSet and its schema with the current DataSet, preserving or discarding changes in the current DataSet.



Best Practices:

- Calling the **AcceptChanges** of the **DataSet** enables you to invoke the method on all subordinates objects (for example, tables and rows) with one call.
- Avoid the MissingSchemaAction property and the FillSchema method as they are slow, because they build the DataSet schema at run time.



Topic: Handling conflicts

Estimated Time: 40 mins.



Objectives: At the end of the activity, the participant will be able to:

- Appreciate the concept of optimistic concurrency.
- Detect possible occurrences of concurrency conflicts.
- Devise ways of resolving conflicts.



Presentation:

Optimistic Concurrency

- Here, database locks are released as soon as data retrieval operations or data update operations are complete.
- Disconnected ADO.NET applications use optimistic concurrency.
- These applications might experience conflicts while updating the data source as the previous value might have been modified by some other application or service while our application was disconnected.

Detecting Conflicts

- The occurrence of errors resulting from concurrency conflicts can be prevented by:
 - Comparing the current data in the database with the original values while performing data modification operations on the database.
 - Both, Data Adapter Configuration Wizard and Command Builder generate such SQL statements that take care of optimistic concurrency violations.

Resolving Conflicts

- Retain the conflicting changes locally in the DataSet so that the database can be updated again later.
- Utilize RowUpdated event of the DataTable for rejecting the conflicting data changes in the local DataSet, DataTable or DataRow.
- Resolve the conflicts later by inspection or programmatically by setting the argument status of the RowUpdated event to UpdateStatus.SkipCurrentRow.
- After the update operation, use the HasErrors property of the DataSet or DataTable's GetChanges().Rows.Count method and DataRow's RowState method for displaying and correcting the conflicts.



Scenario:

Xmart Retail Company has a number of retail stores in many cities. Each Retail store has three counters for billing and maintains one repository for everyday stock. In order to keep the repository updated, the retail store uses an application which implements optimistic concurrency control to display remaining stocks with its prices.



Demonstration/Code Snippet:

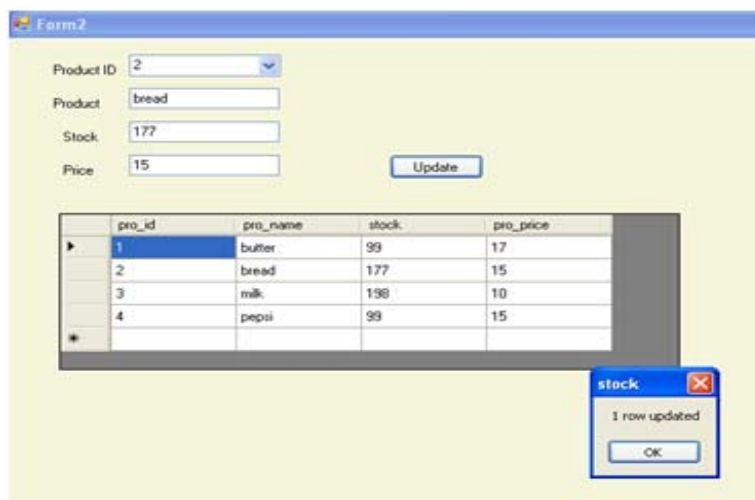


Figure 6.5-1: Output screenshot showing updation of product

Step 1: Open Visual Studio and create a windows application.

Step 2: Create a windows form as shown above. Import the following namespace.

```
using System.Data.SqlClient;

Public Class HandlingConflict:Form

    SqlConnection con;
    SqlCommand prod_cmd;
    SqlDataAdapter prod_da;
    DataSet prod_ds;
```

Step 3: Create a Connection to the Server

```
private void Conflict_Load(object sender, EventArgs e)

    string str;
```

```

str="server=hstslc011;database=Northwind;uid=eltp;pwd=satyam";

con=New SqlConnection();
con.ConnectionString=str;

`Instantiate sqlcommand object to retrieve existing details .Use
sqlDataAdapter to fill the DataSet. Bind the data grid view control to DataSet
for displaying results.

String Cmd_str="select * from stockdetail";
Command=New SqlCommand(cmd_str, con);
prod_da = New SqlDataAdapter();
prod_da.SelectCommand = command;
prod_da.MissingSchemaAction = MissingSchemaAction.AddWithKey;
prod_da.Fill(prod_ds, "stockdetail");
prod_DataGridView.DataSource = prod_ds.Tables(0);

`Instantiate sqlcommand containing update query and add parameters using
sqlparameter class

String cmd_str1="UPDATE stockdetail SET stock=@stock WHERE pro_id=@pro_id";
prod_cmd=New SqlCommand(cmd_str1, con);
prod_cmd.Parameters.Add("@stock", SqlDbType.Int, 6, "stock");
prod_cmd.Parameters.Add("@pro_id", SqlDbType.Int, 6, "pro_id");
prod_cmd.UpdateCommand = command;

}

```

Step 6: The following code fills the textboxes when combobox index gets changed when a new product is selected.

```

Private void ProductID_SelectedIndexChanged(object sender, EventArgs e)
Handles ProductID.SelectedIndexChanged

`Select the details about the selected product
str1 = String.Format("select pro_name,stock,pro_price from stockdetail where
pro_id='{0}'", ProductID.SelectedItem.ToString());
prod_cmd = New SqlCommand(str1, con);
prod_da = New SqlDataAdapter(prod_cmd);
prod_ds = New DataSet;
Try
{

    `Fill the DataSet and display the results as individual items in the
    relevant textboxes.
    Prod_da.Fill(prod_ds);
Catch (Exception ex)
{
    MsgBox(ex.Message)
}
}

```

```
Try
{
    Prod_txt.Text = prod_ds.Tables(0).Rows(0).Item(0).ToString();
    stock_txt.Text = prod_ds.Tables(0).Rows(0).Item(1).ToString();
    price_txt.Text = prod_ds.Tables(0).Rows(0).Item(2).ToString();
    Catch (Exception ex)
    {
        MsgBox(ex.Message);
    }
}
}
```

Step 7: The following code is written in the **Update button** click event. It updates a row with changed values, using a primary key. Find a row from table using primary key column.

```
Private void BtnUpdate_Click(object sender, System.EventArgs e) Handles Button1.Click
```

```
prod_dt=New DataTable;
prod_dt = prod_ds.Tables(0);
prod_dt.PrimaryKey = New DataColumn() {prod_dt.Columns("pro_id")};
int fvalue;
fvalue = CInt(ProductID.SelectedItem.ToString());
DataRow prod_dr= prod_dt.Rows.Find(fvalue);
`Update the datatable into DataSet using update command of sqlDataAdapter class.

Prod_dr("stock") = CInt(stock_txt.Text);
prod_da.Update(prod_ds, "stockdetail");
prod_da.Fill(prod_ds, "stockdetail");
MsgBox("1 row updated ");
```

`The following code uses RowUpdated event of DataAdapter. RecordsAffected property returns a value of 0 when an Exception occurs due to optimistic concurrency violation. SkipCurrentRow ignores the current row and continues the update operation.Add the RowUpdated event handler to any event.

```
AddHandler adap.RowUpdated, New SqlRowUpdatedEventHandler(AddressOf OnRowUpdated);

}
```

Step 8: If the data row is not updated then the following error is thrown.

```
private static void OnRowUpdated(object sender, SqlRowUpdatedEventArgs args)
{
    if ((args.RecordsAffected == 0))
    {
```

```

        args.Row.RowError = "Optimistic Concurrency Violation!";
        args.Status = UpdateStatus.SkipCurrentRow;
    }
}

```

Table used: dbo.nw_federal ; **DataBase:** NorthWind

Table - dbo.stockdetail		Summary	
Column Name	Data Type	Allow Nulls	
pro_id	int	<input type="checkbox"/>	
pro_name	varchar(50)	<input type="checkbox"/>	
stock	int	<input type="checkbox"/>	
pro_price	money	<input type="checkbox"/>	

Figure 6.5-2: Table used in the scenario



Context:

- In environments where a number of users contend for the same data, the probability of concurrency conflicts occurring increases which might lead to unexpected results. The above techniques help in maintaining data consistency.
- Conflict resolution is central to the concept of data management. Without it, working successfully in a networked multi-user environment will be very difficult.



Practice Session:

- E-shoppe is an online retail service where users can bid for products as well as put their products on sale. Prepare an application based on disconnected architecture which takes care of the buying and selling process. Ensure the prevention of any optimistic concurrency violation. For e.g. a product that has already been sold should be removed from the ON SALE list soon after its transaction is complete.



Check list:

- Need of understanding possibilities of concurrency conflicts.
- Various methods which can be employed for resolving conflicts.



Common Errors:

- Not calling and setting the appropriate command (Insert, Update or Delete) before utilizing the Update method of the Data Adapter.
- Using the open () method with the connection object while using DataSet and leaving it unclosed.



Exceptions:

- DBConcurrency Exception might occur if the Insert, Delete or Update command with the Data Adapter resulted in zero rows being affected.

- If the Insert, Delete or Update commands are not set before calling the Data Adapter's update method, InvalidOperationException Exception may occur.
- An Exception might occur if the schema value specified with the MissingSchemaAction property does not exist.



Lessons Learnt:

- Disconnected ADO.NET applications face the problems associated with data consistency as they follow optimistic concurrency whereas connected applications follow Pessimistic Concurrency wherein a database table is locked until an application completes its interaction with it.
- Unlike disconnected ADO.NET applications, connected ADO.NET applications do not allow a number of users to access the database tables at the same time.
- Data Adapter Configuration wizard can be used to add SQL tests to the Insert Command, Update Command or Delete Command for dealing with the conflicts associated with optimistic concurrency.



Best Practices:

- Because the DataSet is, by design, disconnected from the data source, you need to ensure that your application avoids conflicts when multiple clients update data at the data source, according to the optimistic concurrency model.
- There are several techniques when testing for an optimistic concurrency violation.
 - Including a timestamp column in the table.
 - To verify that all the original column values in a row still match those found in the database by testing using a WHERE clause in your SQL statement.
- Develop the code of the application in such a way that the updates are made a part of a transaction which can be rolled back in case one of the update fails.



Topic: Typed and UnTyped DataSet

Estimated Time: 50 mins.



Objectives: At the end of the activity, the participant will be able to:

- Create a Typed DataSet
- Build an UnTyped DataSet
- Understand how to load data into a Typed DataSet



Presentation:

Typed DataSet

- It inherits all the methods, events, and properties of a DataSet and provides strongly typed methods, events, and properties.
- Tables and columns that are part of it can be accessed using user-friendly names and strongly typed variables instead of using collection-based variables.
- Typed DataSets derive their schema (table and column structure) from .xsd files and are easier to program against.
- It provides type checking at compile time.

UnTyped DataSet

- It has no corresponding built-in schema.
- Tables, columns, and so on in an UnTyped DataSet are exposed only as collections.
- DataSet's structure can be created as a schema using the DataSet's WriteXmlSchema method after creating its data members.



Scenario:

Federal Tax Review Committee is an organization which deals with verifying whether all the citizens are regular and honest in terms of their yearly tax payments to the government. They maintain a database which stores vital information about the assets and tax status of all the citizens. The organization uses an application which helps in sorting out the tax defaulters by utilizing typed and UnTyped DataSets.



Demonstration/Code Snippet:

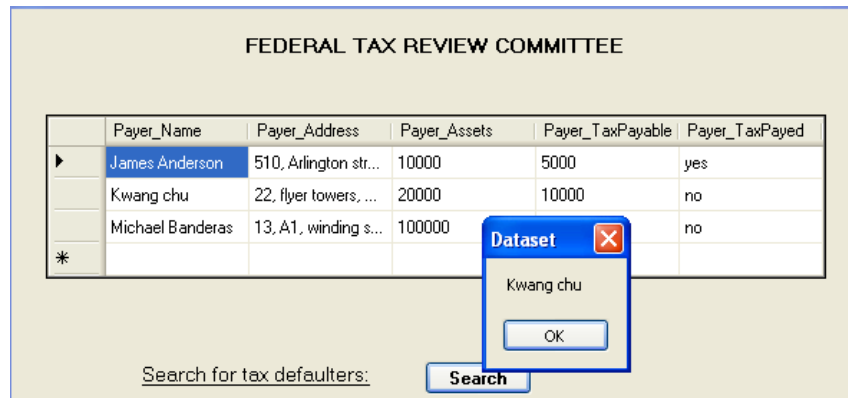


Figure 6.6-1: Output screenshot showing the tax defaulter

Step 1: Generate a XML schema which can be used for utilizing typed DataSet. This can be done either implicitly or explicitly (like in this case).

Step 2: Prepare an XMLFile which contains the schema definition

```
<?xml version="1.0" encoding="utf-8" ?>
<xs:schema id="TaxpayerDataSet" xmlns=""
xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:msdata="urn:schemas-
microsoft-com:xml-msdata">
  <xs:element name="TaxpayerDataSet" msdata:IsDataSet="true">
    'this specifies the name for our typed DataSet'
    <xs:complexType>
      <xs:choice maxOccurs="unbounded">
        <xs:element name="Taxpayers" >
          'this specifies the name which will be used for accessing the DataSet'
          <xs:complexType>
            <xs:sequence>
              <xs:element name="Payer_Name" type="xs:string"
minOccurs="0" />
              'this specifies the name of the column which we will be able to move to
without using the datacolumn collection.'
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:choice>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Step 2: In visual studio's command prompt, input the below command which uses the xsd.exe tool for generating the XSDSchemaFileName.vb file.

```
xsd.exe /d /l: VB XSDSchemaFileName.xsd /n:XSDSchema.Namespace
```



Step 3: Input the following command for compiling the generated code as a library using vb compiler(vb.exe)

```
vbc.exe /t:library XSDSchemaFileName.vb /r:System.dll /r:System.Data.dll
```

Step 4: Create a windows application in Visual Studio and add reference of this library to our windows application so that it can be imported.

Step 5: Import the following namespaces

```
using System.Data;
using System.Data.SqlClient;
using XSDSchema.Namespace;
'XSDSchema.Namespace is our explicitly declared library.
```

Step 6: Establish connection with the data source.

```
Class Form1
{

private void Form1_Load(object sender, System.EventArgs e) Handles MyBase.Load
{

try
{
    SqlConnection myConnection = new
SqlConnection("server=htstslc011;database=northwind;uid=sa;password=satyam");
'Instantiate the sqlDataAdapter class and pass the query string and connection
string into it.
    string dgfill;
    dgfill = "select * from nw_federal";
    SqlCommand dgcmd = new SqlCommand(dgfill, myConnection);
    SqlDataAdapter dgadap = new SqlDataAdapter(dgcmd);
'Use the sqlDataAdapter for filling the DataSet.
    Data.DataSet taxpayers = new Data.DataSet();
    dgadap.Fill(taxpayers);
'Bind the DataGridView control to our data source for displaying the generated
results.
    dgTaxpayerInfo.DataSource = taxpayers.Tables[0];
}
catch (Exception ex)
{
    // handles any unexpected errors
    MsgBox(ex.Message);
}
}
```

Step 6: The below code deals with displaying the names of all those taxpayers who have defaulted their tax payments by using typed DataSet. Begin with establishing connection with the data source.



Business Transformation. Together.

```
private void btnSearch_Click(object sender, System.EventArgs e)
{
    try {
        SqlConnection myConnection = new SqlConnection("server=hstslc011;d
atabase=northwind;uid=sa;password=satyam");
        'Pass the query string and the connection string to the sql data adapter
        string taxstr;
        string search;
        search = "no";
        taxstr = string.Format("select * from nw_federal where Payer_TaxPa
yed='{0}'\'", search);
        SqlCommand taxcmd = new SqlCommand(taxstr, myConnection);
        SqlDataAdapter taxadap = new SqlDataAdapter(taxcmd);

        'Fill the typed DataSet with the generated results
        XSDSchema.Namespace.TaxpayerDataSet taxpayers = new XSDSchema.Name
space.TaxpayerDataSet();
        taxpayers.BeginLoadData();

        'begins the loading of data into the typed DataSet
        taxadap.Fill(taxpayers.Taxpayers);
        taxpayers.EndLoadData();

        'ends the loading of data into the typed DataSet
        'As taxpayers is a typed DataSet, it is strongly named. Hence, we
can access the rows of the generated results by utilizing the
TaxpayersRow method of our UnTyped DataSet.
        XSDSchema.Namespace.TaxpayerDataSet.TaxpayersRow taxpayerrow;

        'Display the name of the defaulters one after the other
        foreach (taxpayerrow in taxpayers.Taxpayers)
        {
            MsgBox(taxpayerrow.Payer_Name);

            'Here we are accessing the payer_name column without using the
DataColumnCollection method of DataSets. If we would have specified any
other column name in the XML schema we defined earlier, we would have
been able to access that column similarly.
        }
    }
    catch (Exception ex)
    {
        MsgBox(ex.Message);
    }
}
```

Tables used: dbo.nw_federal ; **DataBase:** NorthWind

Table - dbo.nw_federal	Table - dbo.nw_federal	Summary
Column Name	Data Type	Allow Nulls
Payer_Name	varchar(50)	<input type="checkbox"/>
Payer_Address	varchar(500)	<input type="checkbox"/>
Payer_Assets	int	<input type="checkbox"/>
Payer_TaxPayable	int	<input type="checkbox"/>
Payer_TaxPaid	varchar(50)	<input type="checkbox"/>

Figure 6.6-2: Table used in the above scenario



Context:

- An UnTyped DataSet is useful when no schema is available for the DataSet and when the data does not have a static, predictable structure.
- Typed DataSet provides type checking at compile time which reduces error possibilities.
- Access to tables and columns in a typed DataSet is determined at compile time, not through collections at run time.



Practice Session:

- Mr. Brown looks after all the ERP related activities of AUSTERE Mfg. Company Ltd. The work primarily involves tracking the status of the orders placed by the company for raw materials. Prepare a track sheet containing ordered, date of placing the order, delivery date, amount paid, amount due, and product category ID. Use typed DataSet for building the application.



Check list:

- Implementation of Typed DataSet through an XML schema.
- Implementation of UnTyped DataSet.



Common Errors:

- Incorrect information being specified in the connection string and query string.
- Incorrect names being provided while developing the XML schema definition.
- Not compiling the schema as a library before importing it.



Exceptions:

- An ArgumentException might occur if the name and DataType specified in our schema does not match with the table structure in the database.
- FormatException might occur if a DataError event gets raised due to the presence of an error

while committing data to the data source.



Lessons Learnt:

- Aside from the improved readability of the code, a typed DataSet also allows the Visual Studio .NET code editor to automatically complete lines as you type.
- With a strongly typed DataSet, type mismatch errors are caught when the code is compiled rather than at run time.
- You can use either typed or UnTyped DataSets in your applications. However, Visual Studio has more tool support for typed DataSets, and programming with them is easier and less error-prone.
- If you change the name of a column in your DataSet and then compile your application, you receive a build error.

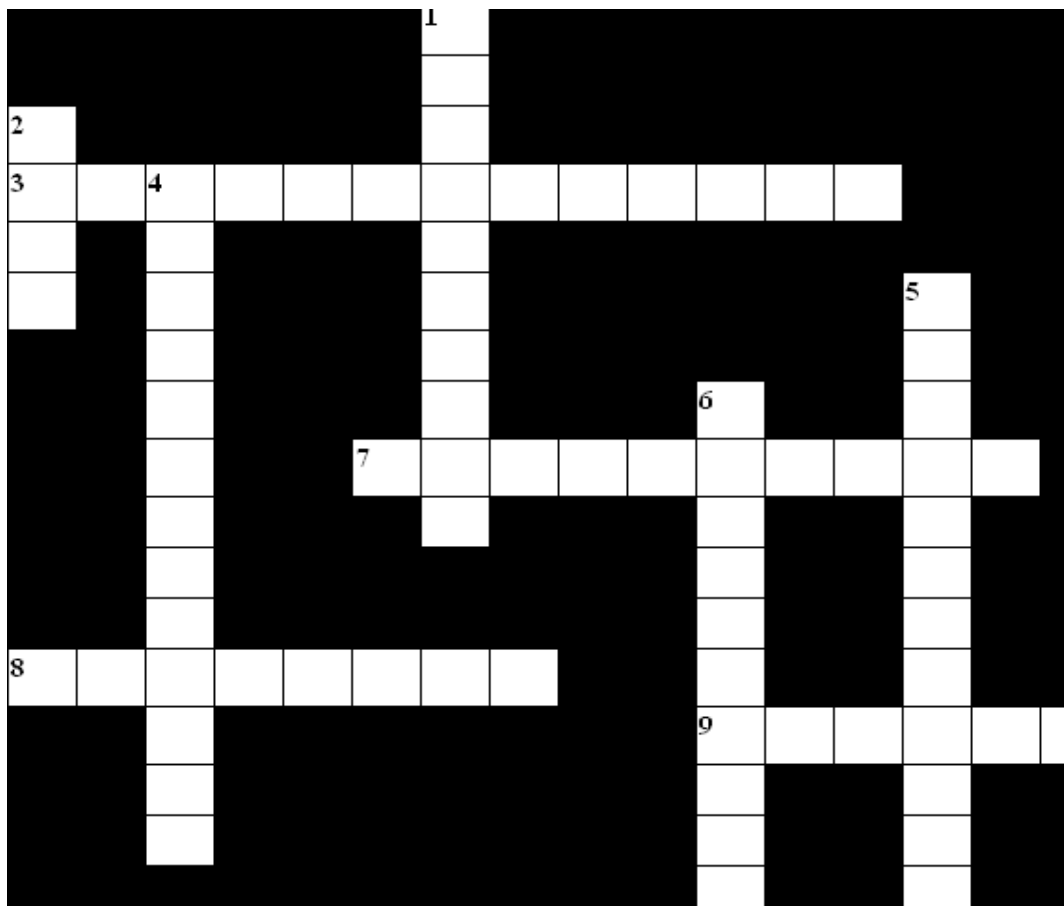


Best Practices:

- When you have fixed schema or relational structure for your DataSet, you can create a strongly typed DataSet that exposes rows and columns as properties of an object rather than items in a collection. For example, instead of exposing the name column of a row from a table of customers, you expose a Customer object's Name property.
- A typed DataSet derives from the DataSet class, so that you do not sacrifice any of the DataSet functionality. That is, a typed DataSet can still be remoted and can be supplied as the data source of a data-bound control such as a DataGrid
- When using a strongly typed DataSet, you can annotate the XML Schema definition language (XSD) schema of the DataSet to ensure that your strongly typed DataSet appropriately handles null references.

Crossword: Unit-6

Estimated Time: 10 mins.



Across:

3. DataAdapter provides a property that represent database commands(13)
7. A property of datatable used to display the conflicting rows(10).
8. a call to this method, the row is removed immediately. Any corresponding rows in the back end data source will not be affected(8) .
9. The method of the DataAdapter is called to resolve changes from a DataSet back to the data source(6)

Down:

1. If an error occurs (a conflict), all the remaining rows updating process is aborted. You need to control this by trapping these errors in a Try/Catch block.The property used by datatable is(10).
2. A method of DataAdapter that ensures the consistent data update into the datasource(4).
4. DataAdapter provides a property that represent database commands(13)
5. It is a DataSet that is first derived from the base DataSet class and then uses information in an XML (12)
6. Using which property of a DataGrid control, you can fill various kinds of data in a DataGrid(10) .

Answers for Crosswords

Unit-1

Across	1) DataSet 3) Update 4)Open 6)Constraint
Down	2) Executenonquery 5) Dataview

Unit-2

Across	1) Dispose 3)DataAdapter 5) Webconfig 6) Transactionscope 7) Leak
Down	1) DataSet 2) Datareader 4) Zero

Unit-3

Across	3) Command 4) Datareader 7) Storedprocedure 8) Save
Down	1) Connected 2) DataAdapter 5) Executescalar 6) Commit

Unit-4

Across	1) Add 2) Getchanges 3) Rejectchanges 5) dataview 6) Typed
Down	1) Acceptchanges 2) Removeit 4) Databinding

Unit-5

Across	2) Keyset 3) DataAdapter 6) Getchanges 7) TypedDataSet
Down	1) Dynamic 2) DataSet 4) XSD

Unit-6

Across	3) Insertcommand 7)Getchanges 8) Removeat 9) Update
Down	1) Haschanges 2) Fill 4) Selectcommand 5) TypedDataSet 6) Datasource

....

Team Members



Srikanth Nivarthi
47275



Sreenivas Ram
66223



Seshu Babu Barma
56150



Veerendra Kumar Ankem
77964



Teena Arora
74572

Contributors



Subbulaxmi R.

66808



Swaroop Kavali

72139



Naiya Desai

72127



Chaitanya Damley

72229



Surekha Vullekki

68576



Prateek Gokhale

72157



RamaChandu Immidisetty

72098