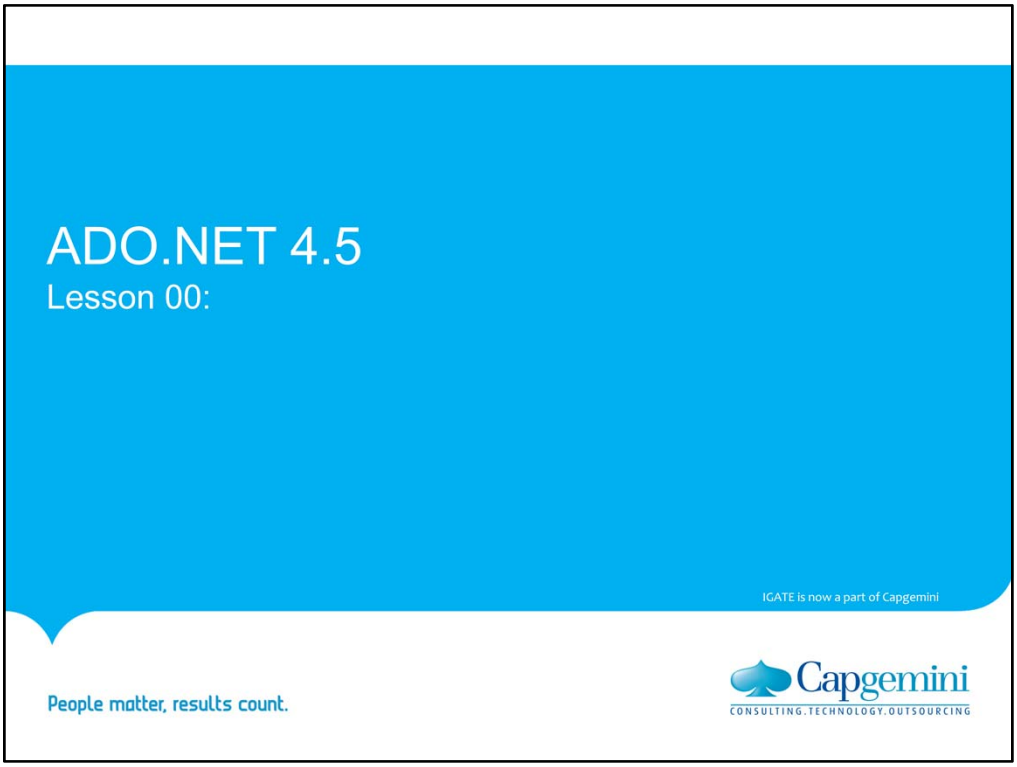


Document History

Date	Course Version No.	Software Version No.	Developer / SME	Change Record Remarks
18-Aug-09	1	.NET 3.5	Anu Mitra	
31-Oct-09	1	.NET 3.5	Documentation Team	Quality Review
25-July-2011	2	.NET 3.5	Shilpa Bhosle	Changes in material made based on integration process
13-Feb-2015	3	.NET 4.5	Nachiket Inamdar	Incorporated ADO.NET 4.5 Features

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Course Goals and Non Goals

- Course Goals
  - Developing Windows Application in .NET using ADO.net
  - XML Support in ADO.NET 4.5
  - Implementing Layering Approach in ADO.NET 4.5
- Course Non Goals
  - Implementation of ADO.NET 4.5 in Web Based Applications

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 3

Pre-requisites

- C# 5.0
- XML
- Windows Forms
- Basic knowledge of T-SQL Queries

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 4

Intended Audience

- .NET Application Developers

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 5

Day Wise Schedule

- Day 1
  - Lesson 1: Introduction to ADO.NET 4.5
  - Lesson 2: Working with Connected Architecture
- Day 2
  - Lesson 3:Working with disconnected architecture
  - Lesson 4: XML support in ADO.NET
- Day 3
  - Lesson 5: Asynchronous Data Access
  - Lesson 6: Implementing Data Access Layer in ADO.NET

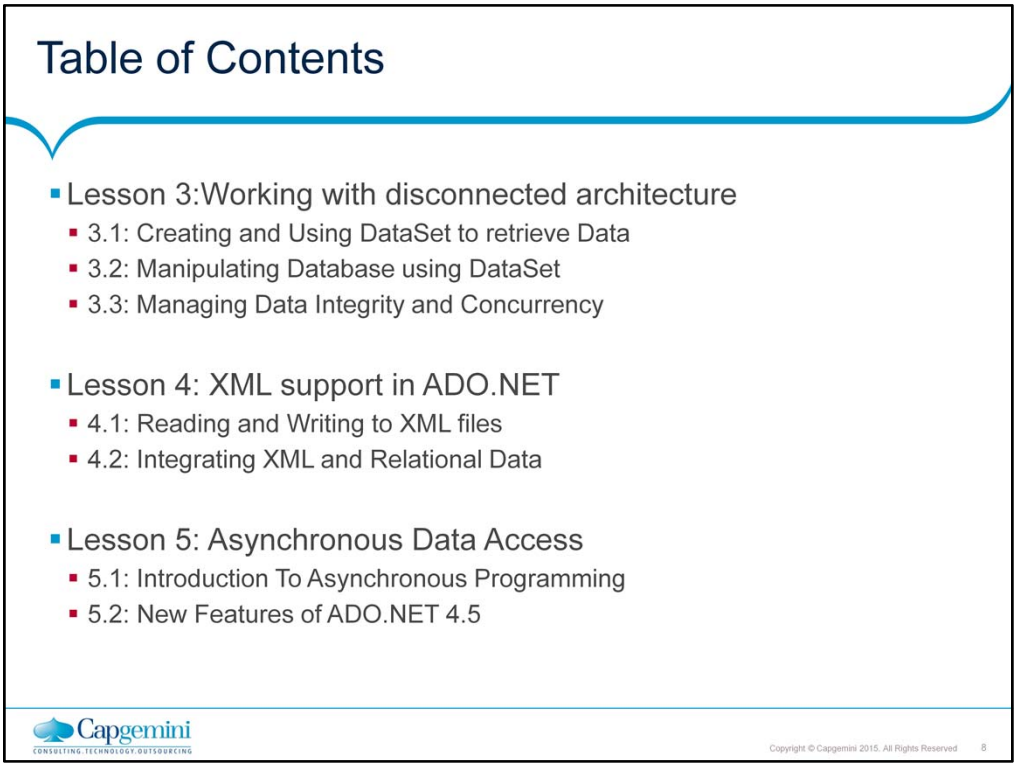
Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 6

Table of Contents

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 7



Lesson 2: Querying and Modifying Data Using ADO.Net Commands

- 2.1. Commands to Retrieve Data
- 2.2. Commands to Manipulate Data
- 2.3. Managing Data Integrity and Concurrency

Table of Contents

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 9

References

- ADO.NET 4.5 Cookbook - O'Reilly
- ADO.NET In Nutshell - O'Reilly
- MSDN

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 10

Next Step Courses (if applicable)

- LINQ
- Entity Framework

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 11

Other Parallel Technology Areas

- JDBC

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 12

ADO.NET 4.5

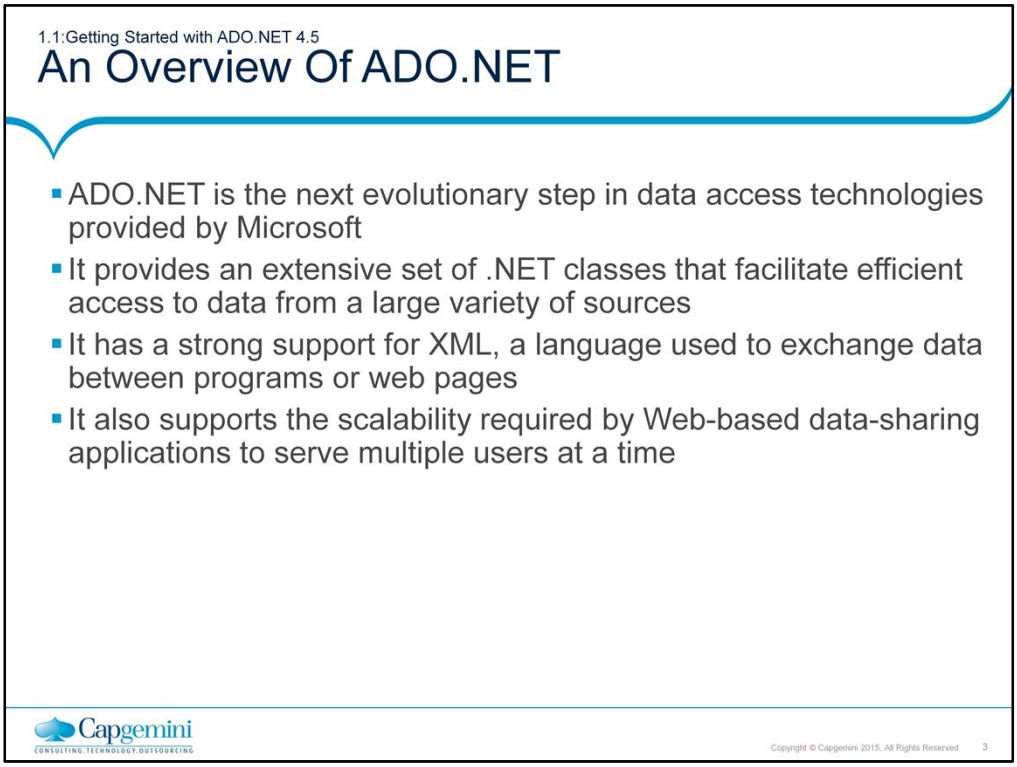
Lesson 1: Introduction to ADO.NET 4.5

Lesson Objectives

- In this lesson, you will learn about:
  - Getting started with ADO.NET 4.5
  - Overview of .NET Data Providers
  - ADO.NET Connected and Disconnected Architecture
  - ADO.NET Generic Classes
  - Best Practices for Managing Connections

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 2



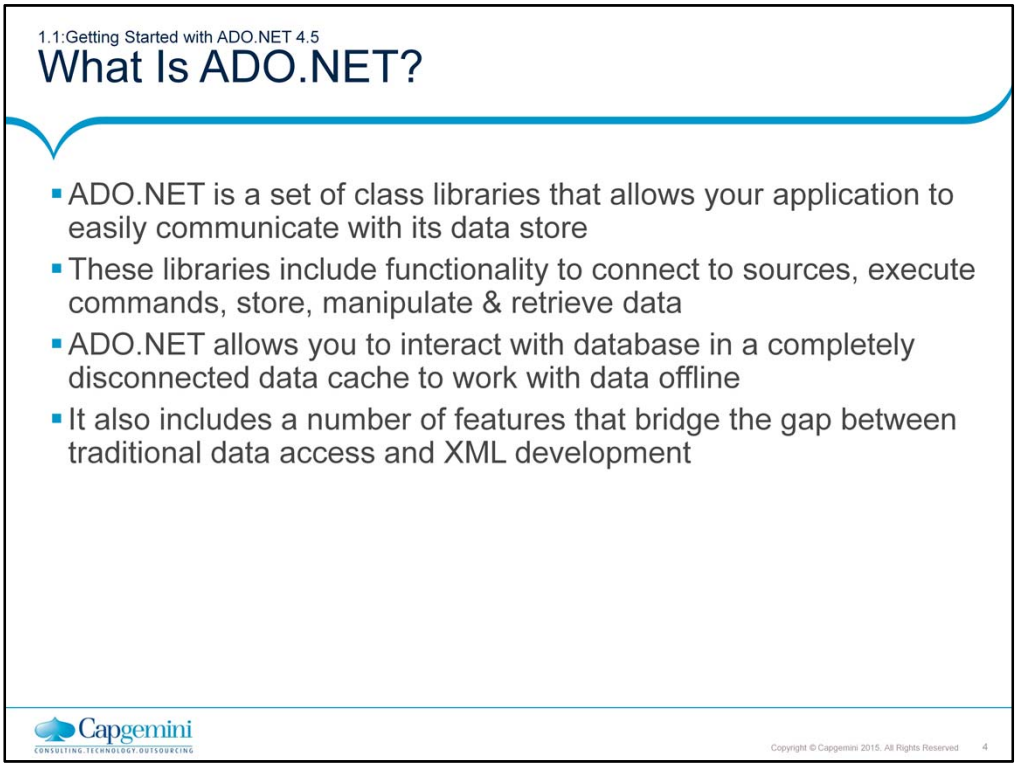
An overview of ADO.NET

ADO.NET is Microsoft's latest data access technology. As an integral part of the .NET Framework it is far more than simply an upgrade of previous incarnations of ActiveX Data Objects ADO. ADO.NET provides an extensive set of .NET classes that facilitate efficient access to data from a large variety of sources also enabling sophisticated manipulation and sorting of data. ADO.NET is essentially a collection of classes that expose methods and attributes used to manage communications between an application and a data store.

ADO.NET is essentially a collection of classes that expose methods used to manage communications between an application and a data store. Being an integral part of the .NET Framework, ADO.NET simplifies integration of data sharing in distributed ASP.NET applications.

Native support for XML is another principal feature for ADO.NET. ADO.NET enables you to create an XML representation of a dataset, with or without its schema. In an XML representation the data is represented in the XML format, and the data metadata is written by using the XML Schema definition language (XSD). XML and XML Schema provide a convenient format for transferring the contents of a actual data to and from remote clients.

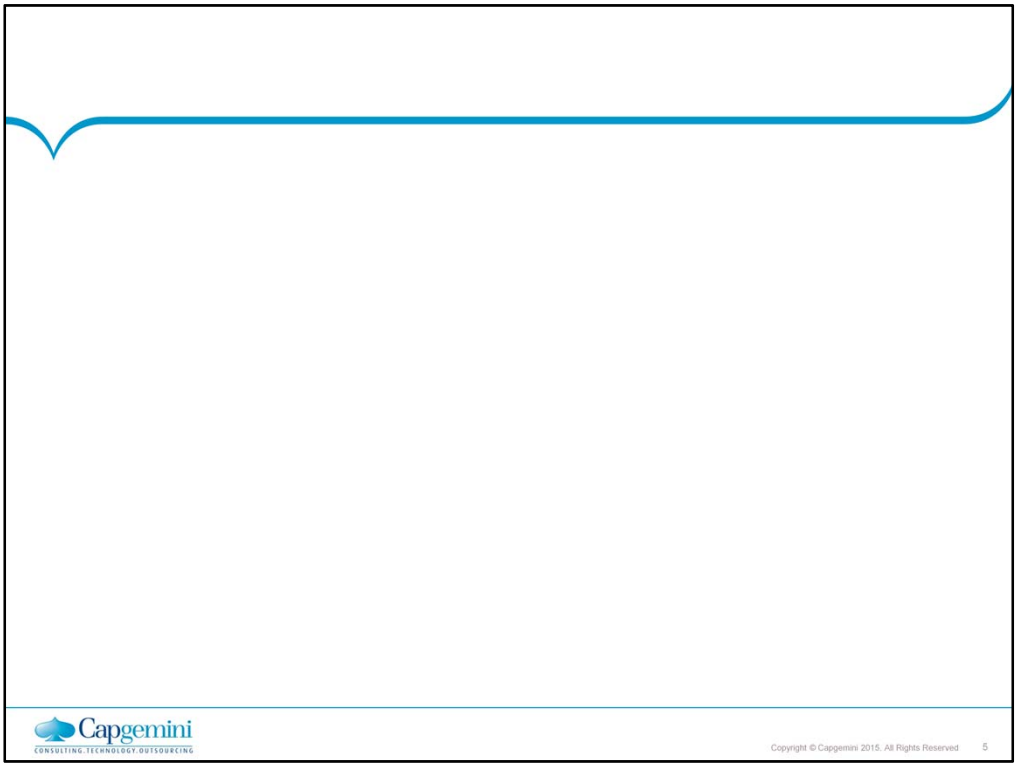
ADO.NET also supports the scalability required by Web-based data-sharing applications. Web applications must serve tens, hundreds, or even thousands of users. ADO.NET does not retain lengthy database locks or active connections that monopolize limited resources. This allows the number of users to grow with only minimal increase in the demands on the resources of a system.



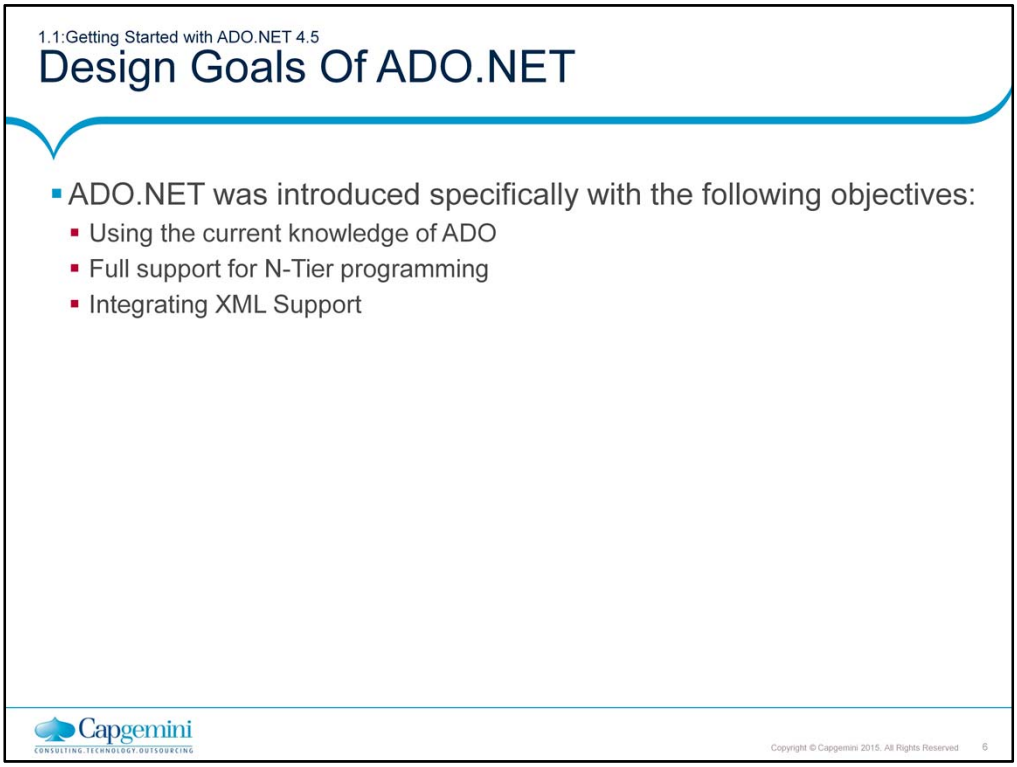
What is ADO.NET?

Nearly all business applications need to store large volumes of data and storage of data is accomplished by a database management system. Accessing data has become a major programming task for modern software programming, both for standalone applications and for web applications. A mechanism is needed to work with this data without having to concern low level details such as how data is stored in a database.

Microsoft's ADO.NET technology offers a solution to many of the problems associated with data access. ADO.NET is a .NET library including a set of classes which expose data access services to any .NET programmer. With ADO.NET you get a rich set of components for creating distributed, data-sharing applications. It is an integral part of the .NET Framework, providing access to relational, XML, and application data. ADO.NET supports a variety of development needs, including the creation of front-end database clients and middle-tier business objects used by applications, tools, languages, or Internet browsers. ADO.NET provides consistent access to data sources such as Microsoft SQL Server and XML, and also to data sources exposed through OLE DB and ODBC.



Data-sharing consumer applications can use ADO.NET to connect to these data sources and retrieve, manipulate, and update the data that they contain. ADO.NET separates data access from data manipulation into discrete components that can be used separately or in sequence. ADO.NET includes .NET Framework data providers which can be used for connecting to a database, executing commands, and retrieving results. Microsoft's ADO.NET announced the introduction of a disconnected mode of data access, enabling data exchange even across process boundaries efficiently. This is in sharp contrast to the earlier data access technologies with only connected data access mode of operation. It should be noted that ADO.NET supports both connected and disconnected mode of data access. We will be discussing more on the disconnected feature as we go along the course.



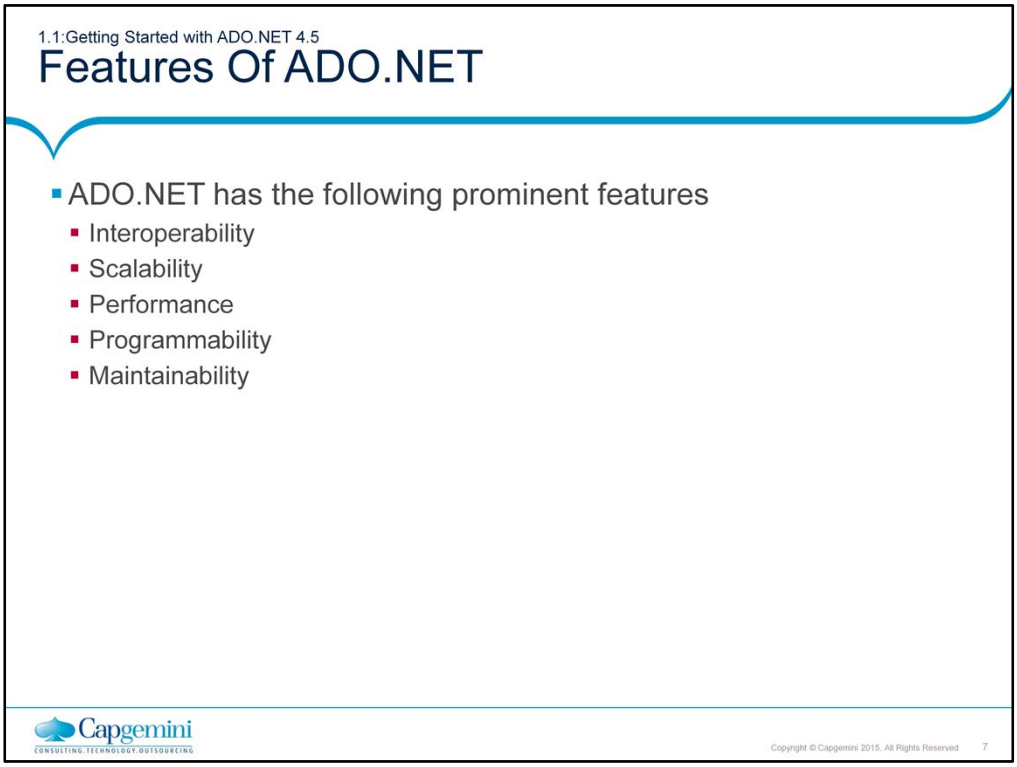
Design Goals of ADO.NET
ADO.NET was designed to meet certain needs :

Using the current knowledge of ADO: The design for ADO.NET addresses many of the today's application development model requirements. However, at the same time, the programming model stays as similar as possible to ADO, so that the current ADO developers do not have to start from the scratch. ADO.NET is an intrinsic part of the .NET Framework yet retains familiarity to the ADO programmer.

ADO.NET also coexists with ADO. Although most new .NET-based applications will be written using ADO.NET, ADO remains available to the .NET programmer through .NET COM interoperability services.

Supporting the N-Tier Programming Model: The concept of working with a disconnected set of data has become a focal point in the programming model. ADO.NET provides first-class support for the disconnected, n-tier programming environment for which many new applications are written.

Integrating XML Support: XML and data access are closely tied. XML is about encoding data, and data access is increasingly becoming about XML. XML support is built into ADO.NET at a very fundamental level. The XML classes in the .NET Framework and ADO.NET are part of the same architecture; they integrate at many different levels. You therefore no longer have to choose between the data access set of services and their XML counterparts; the ability to cross over from one to the other is inherent in the design of both.

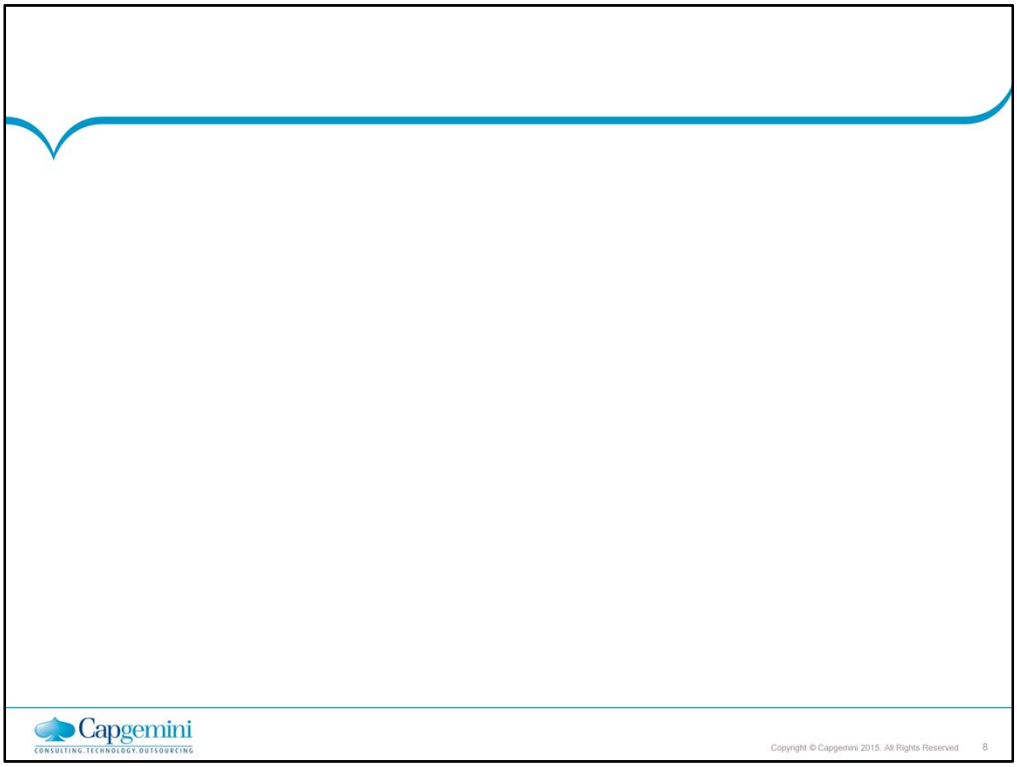


Features of ADO.NET:
ADO.NET offers several features which are advantageous over the previous data access methods. They can be broadly categorized as:

Interoperability: All data in ADO.NET is transported in XML format. In other words ADO.NET applications takes full advantage of the flexibility and broad acceptance of XML. An ADO.NET application can transmit data in XML format and at the receiving end any component which can be a Visual Studio application or any other application can receive the data. The only requirement is that the receiving component should be able to read XML. As an industry standard, XML was designed keeping this kind of interoperability in mind.

Scalability: Today's implementation model of applications can have huge data requirements, and hence, scalability is necessary. The client/server model is out which had limited amount of requirements. An application that consumes resources such as database locks and database connections may serve hundreds of users well, but as the number increases, it will not continue to do so. ADO.NET promotes the use of disconnected datasets, with automatic connection pooling bundled as part of the package which accommodates scalability.

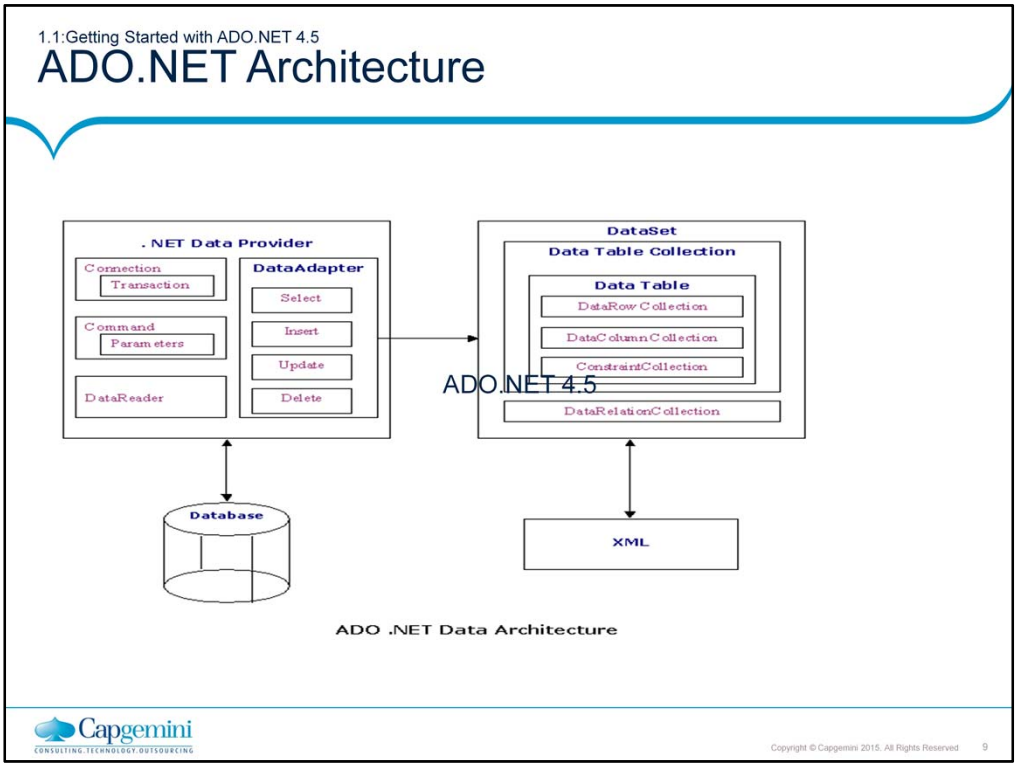
Performance: Since ADO.NET is mainly about disconnected datasets, the database server is no longer a bottleneck, and hence, applications should incur a performance boost. While transmitting a disconnected recordset among tiers, a significant processing cost can result from converting the values in the recordset to data types. In ADO.NET, such data type conversion is not necessary.



Features of ADO.NET:

Programmability - ADO.NET data components in Visual Studio encapsulate data access functionality in various ways that help you program more quickly and with fewer mistakes. For example, data commands abstract the task of building and executing SQL statements or stored procedures.

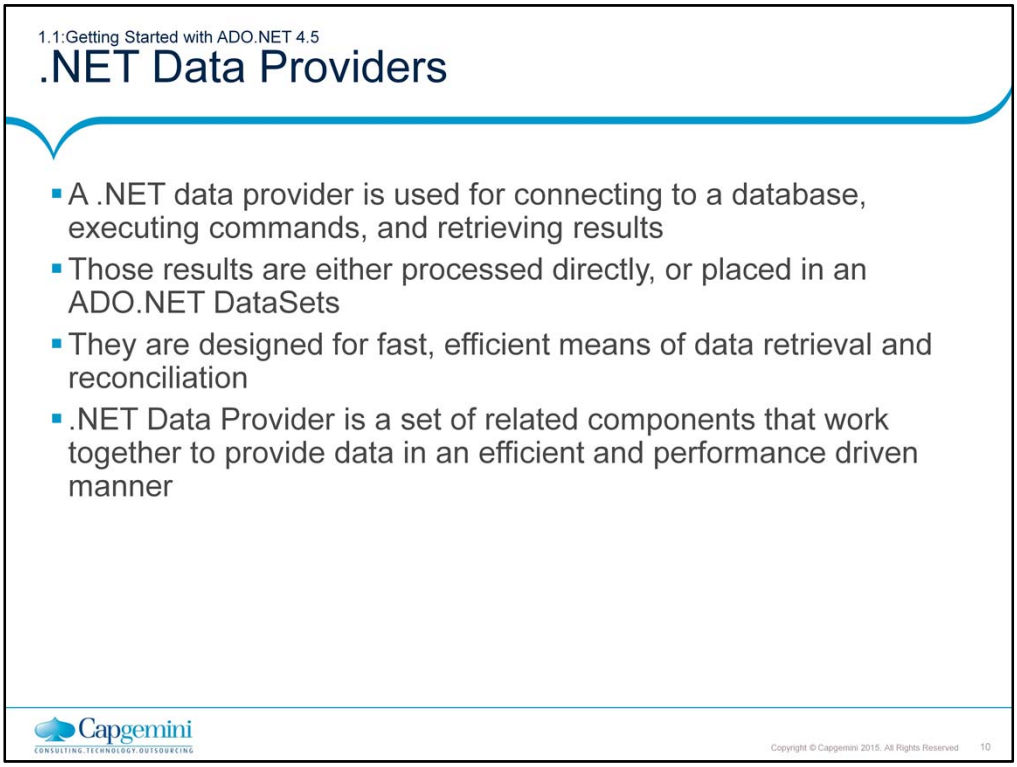
Maintainability: In the life of a deployed system, modest changes are possible, but substantial, architectural changes are rarely attempted because they are so difficult. That is unfortunate, because in a natural course of events, such substantial changes can become necessary. For example, as a deployed application becomes popular with users, the increased performance load might require architectural changes. As the performance load on a deployed application server grows, system resources can become scarce and response time or throughput can suffer. Faced with this problem, software architects can choose to divide the server's business-logic processing and user-interface processing onto separate tiers on separate machines. In effect, the application server tier is replaced with two tiers, alleviating the shortage of system resources. The problem is not about designing a three-tiered application. Rather, it is about increasing the number of tiers after an application is deployed. If the original application is implemented in ADO.NET using datasets, this transformation is made easier. Remember, when you replace a single tier with two tiers, you arrange for those two tiers to trade information. Because the tiers can transmit data through XML-formatted datasets, the communication is relatively easy.



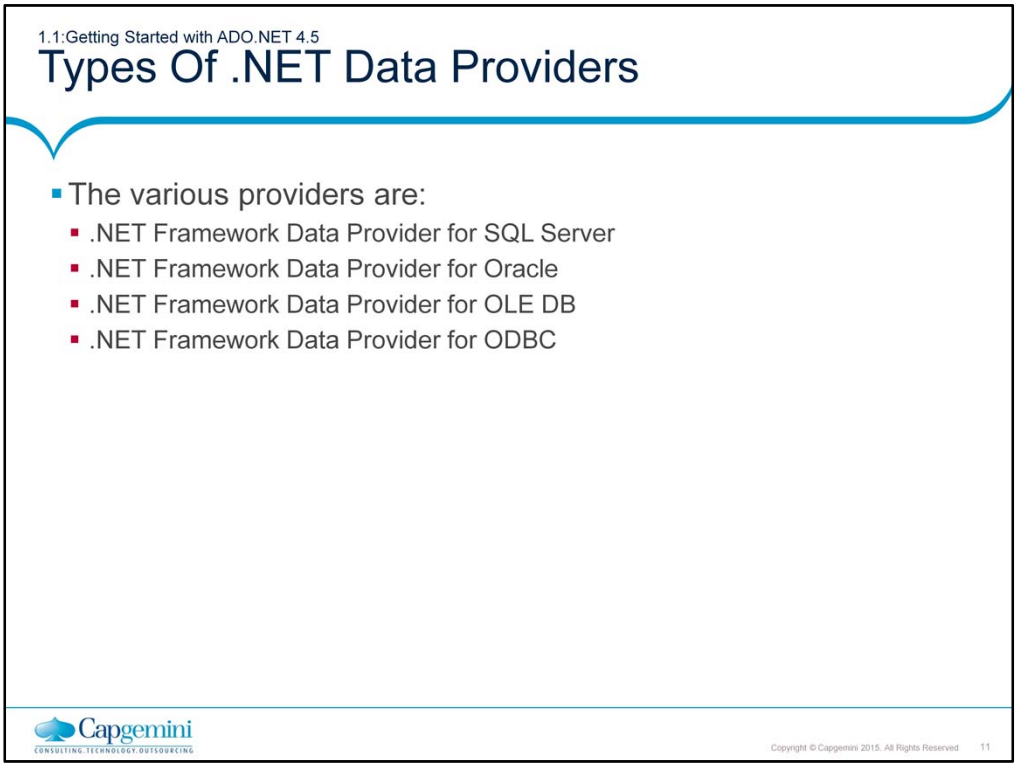
The ADO.NET Architecture :

Like any other architecture, there are certain important parts that make up ADO.NET. In this section, you'll look at the various objects that make up ADO.NET. As you probably know, .NET classes can be grouped under namespaces. All ADO.NET-related functionality appears under the System.Data namespace. This doesn't mean that some other software developer cannot write libraries that don't belong to that namespace, but as the Microsoft .NET Framework ships, all ADO.NET-related functionality sits inside the System.Data namespace. Let's further examine the various details of ADO.NET Architecture.

Data Access in ADO.NET relies on two components - .NET Data Providers and DataSet



The .NET Data Provider is responsible for providing and maintaining the connection to the database. It is a set of related components that work together to provide data in an efficient and performance driven manner.



Data Provider for SQL Server: The .NET Framework Data Provider for SQL Server uses its own protocol to communicate with SQL Server. It is lightweight and performs well because it is optimized to access a SQL Server directly without adding an OLE DB or Open Database Connectivity (ODBC) layer. It provides data access for Microsoft SQL Server version 7.0 or later. The classes for this provider are located in the System.Data.SqlClient namespace. The .NET Framework Data Provider for SQL Server supports both local and distributed transactions. To include this namespace in your application:

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

Data Provider for OLE DB: OleDbDataProvider allows us to connect to other types of databases like Access and Oracle. To use this data provider, include the following namespace in your application:

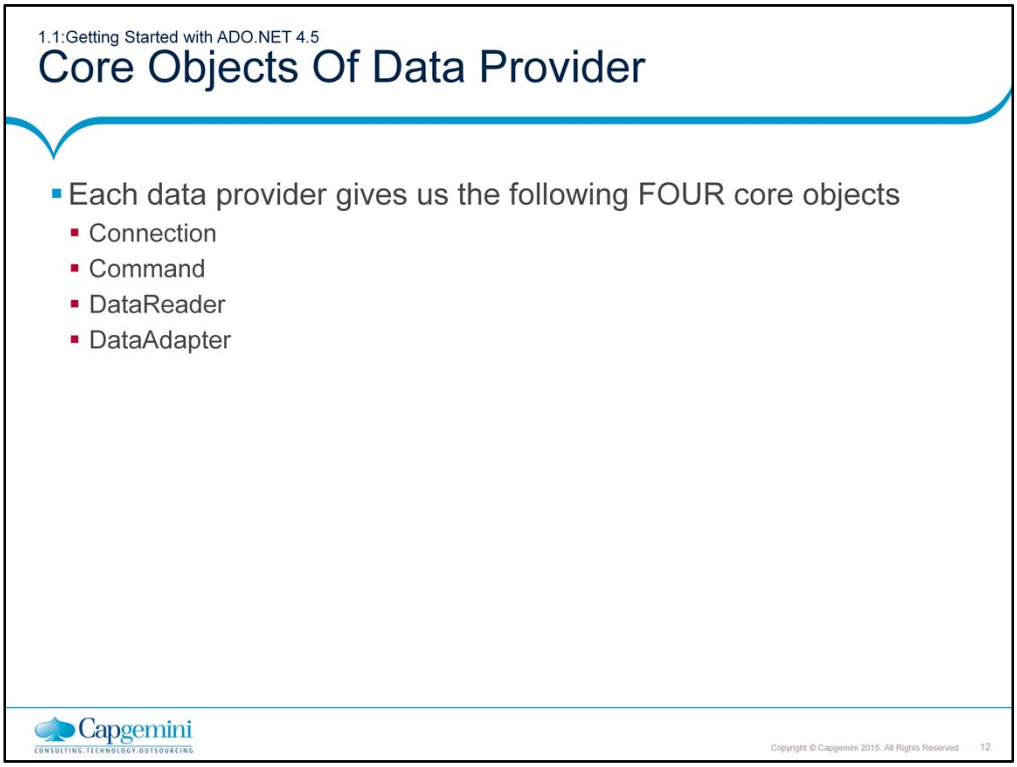
```
using System.Data.OleDb;
```

Data Provider for Oracle: You can use this provider for connecting to Oracle data sources. The .NET Framework Data Provider for Oracle supports Oracle client software version 8.1.7 and later. To use this data provider, include the following namespace in your application:

```
using System.Data.OracleClient;
```

Data Provider for ODBC: You can use this provider for connecting to databases like IBM DB2, Informix etc. To use this data provider, include the following namespace in your application:

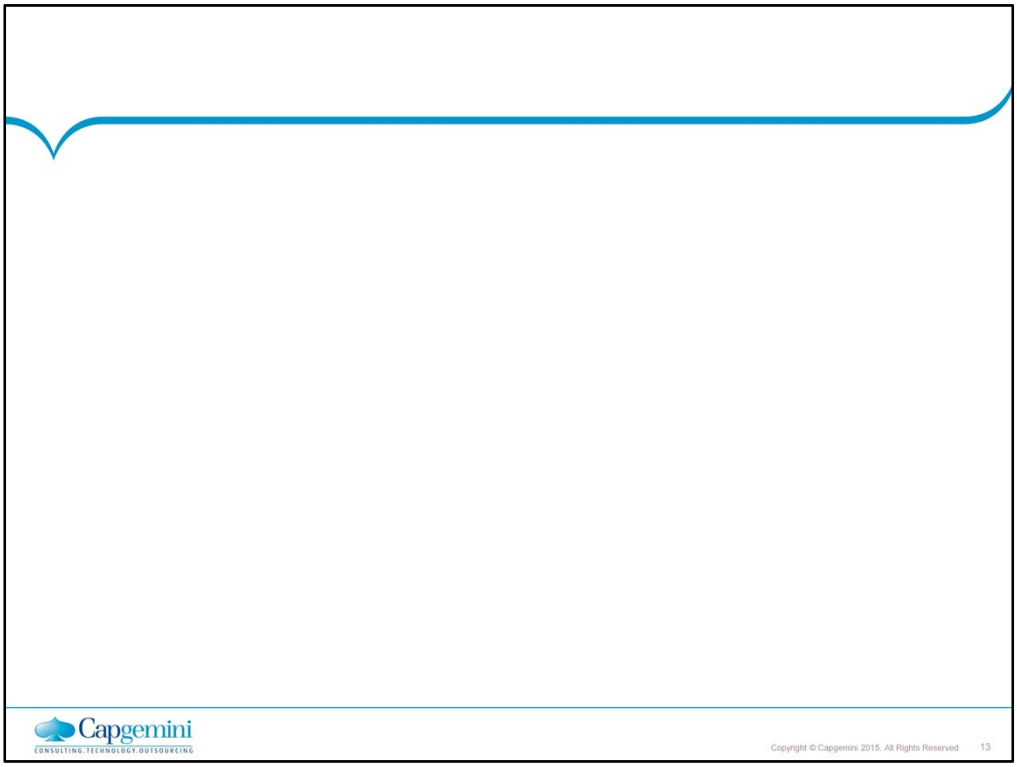
```
using System.Data.Odbc;
```



Core Objects of Data Provider:

Connection: This is the object that allows you to establish a connection with the data source. Depending on the actual .NET data provider involved, connection objects automatically pool physical database connections for you. It's important to realize that they don't pool connection object instances, but they try and recycle physical database connections. Examples of connection objects are OleDbConnection, SqlConnection, OracleConnection, and so on.

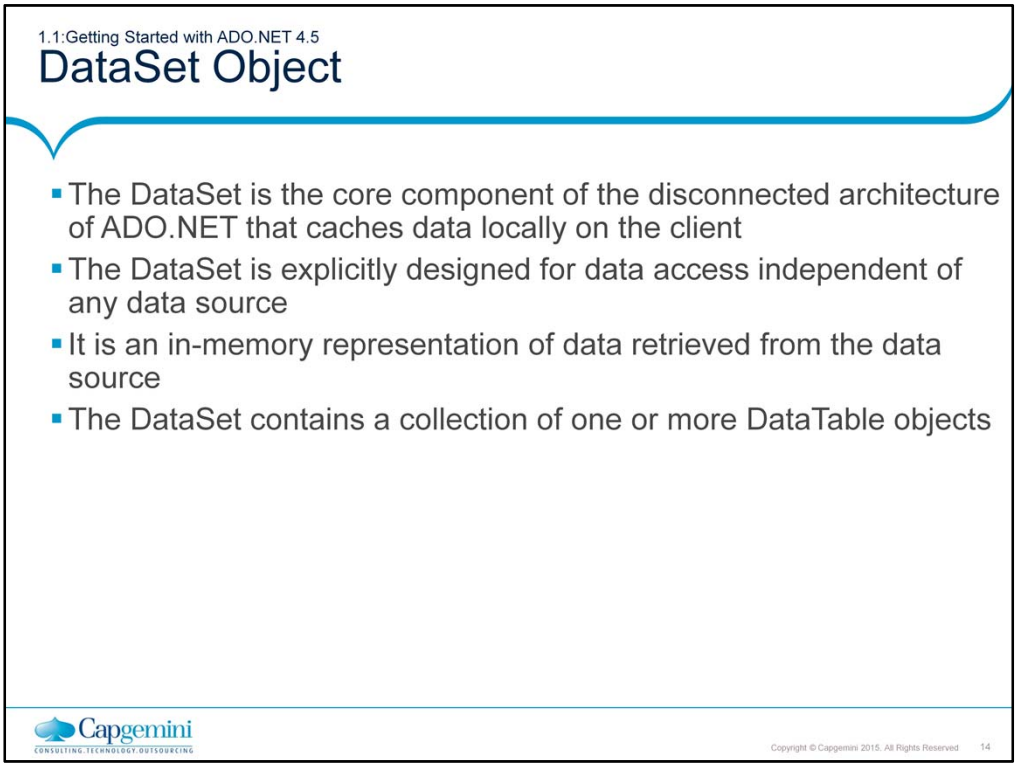
• Command: This object represents an executable command on the underlying data source. This command may or may not return any results. These commands can be used to manipulate existing data, query existing data, and update or even delete existing data. In addition, these commands can be used to manipulate underlying table structures. Examples of command objects are SqlCommand, OracleCommand, and so on. A command needs to be able to accept parameters. The Parameter object of ADO.NET allows commands to be more flexible and accept input values and act accordingly.



Core Objects of Data Provider:

DataReader – This object is designed to help you retrieve and examine the rows returned by your query as quickly as possible. You can use the DataReader object to examine the results of a query one row at a time. When you move forward to the next row, the contents of the previous row are discarded. The DataReader doesn't support updating. The data returned by the DataReader is read-only. Because the DataReader object supports such a minimal set of features, it's extremely fast and lightweight. The disadvantage of using a DataReader object is that it requires an open database connection and increases network activity.

DataAdapter: This object acts as a gateway between the disconnected and connected flavors of ADO.NET. The architecture of ADO.NET, in which connection must be opened to access the data retrieved from database is called as connected architecture where as in a disconnected architecture data retrieved from database can be accessed by holding it in a memory with the help of DataSet object even when connection to database was closed (We will be discussing more about both of these architecture and the objects involved in it in detail on the coming lessons). It establishes the connection for you or, given an established connection, it has enough information specified to itself to enable it to understand a disconnected object's data and act upon the database in a prespecified manner. Examples of DataAdapters are SqlDataAdapter, OracleDataAdapter, and so on. It has commands like select, insert, update and delete. Select command is used to retrieve data from database and insert, update and delete commands are used to send changes to the data in dataset to database. It needs a connection to transfer the data.

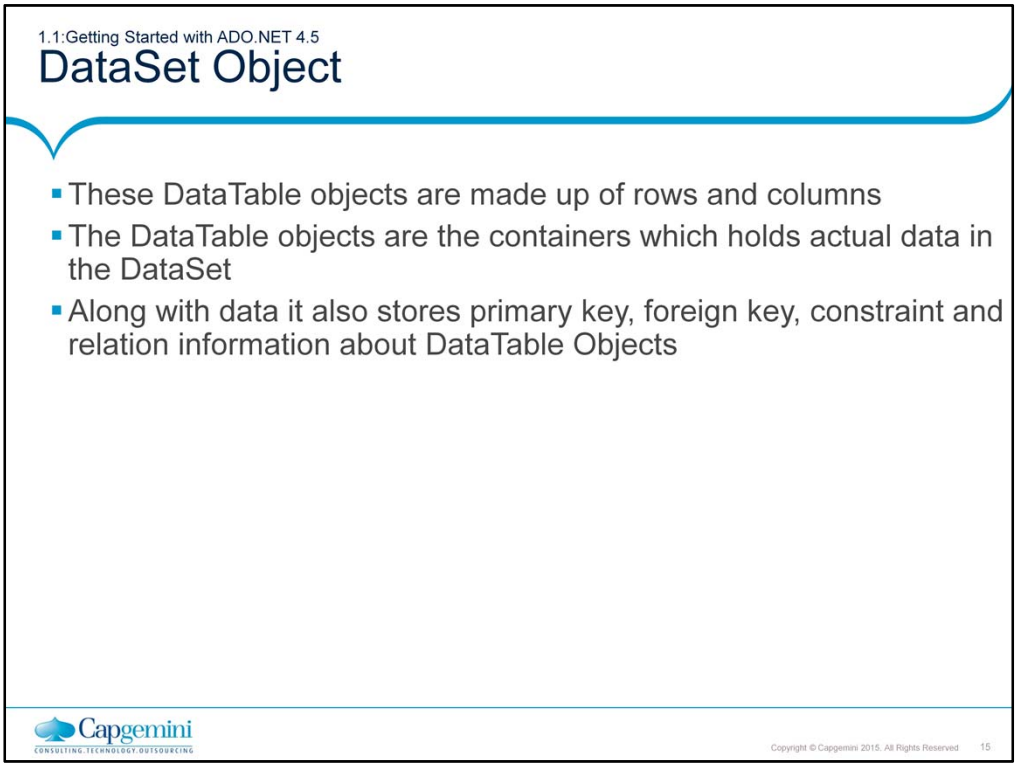


DataSet is one of the very important component of ADO.NET Architecture which basically an independent object is explicitly designed for data access independent of any data source.

DataSet Object – Its an in-memory cache of data retrieved from the data source. The DataSet exhibits similar properties to an in-memory relational database - for example, data is organized into multiple tables using DataTable objects, tables can be related using DataRelation objects, and data integrity can be enforced using the constraint objects UniqueConstraint and ForeignKeyConstraint.

Another feature of the DataSet is that it tracks changes that are made to the data it holds before updating the source data. DataSet are also fully XML-featured. They contain methods such as GetXml and WriteXml that respectively produce and consume XML data easily. In an XML scenario where there is no database, these methods enable use of ADO.NET without the Data Provider being involved.

The DataSet object provides a consistent programming model that works with all current models of data storage: flat, relational, and hierarchical. It represents the data that it holds as collections and data types. The data within a DataSet is manipulated via the set of standard APIs exposed through the DataSet and its child objects regardless of its data source.



The DataSet Object in detail

DataTable: A DataSet object is made up of a collection of tables, relationships, and constraints. In ADO.NET, DataTable objects are used to represent the tables in a DataSet object.

DataColumn: Each DataTable object has a Columns collection, which is a container for DataColumn objects. A DataColumn object corresponds to a column in a table.

DataRow: To access the actual values stored in a DataTable object, use the object's Rows collection, which contains a series of DataRow objects.

DataView: Once we've retrieved the results of a query into a DataTable object, we can use a DataView object to view the data in different ways. If we want to sort the contents of a DataTable object based on a column, simply set the DataView object's Sort property to the name of that column. We can also use the Filter property of the DataView so that only the rows that match certain criteria are visible. We can use multiple DataView objects to examine the same DataTable at the same time.

DataRelation: A DataSet, like a database, might contain various interrelated tables. A DataRelation object lets you specify relations between various tables that allow you to both validate data across tables and browse parent and child rows in various DataTables.

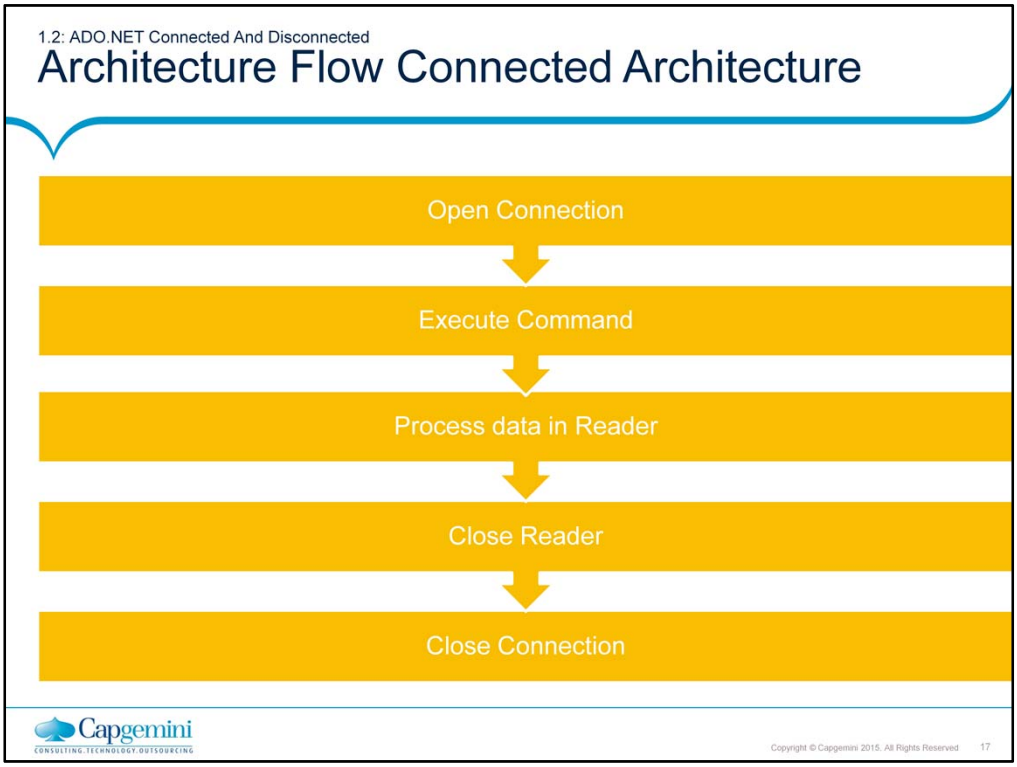
1.1:Getting Started with ADO.NET 4.5

ADO.NET Connected Architecture

- In Connected Architecture the application makes a connection to the Data Source
- It interacts with data source through SQL requests using the same connection and command objects
- After the execution of command, resultset is fetched and used for read operations
- Any kind of updates takes place directly in the data source
- Disadvantages of Connected environment are:
  - Database Locking issue
  - Required constant database connection

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 16


1.2: ADO.NET Connected And Disconnected
Architecture Flow Connected Architecture
Open Connection
Execute Command
Process data in Reader
Close Reader
Close Connection
Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING
Copyright © Capgemini 2015. All Rights Reserved
17

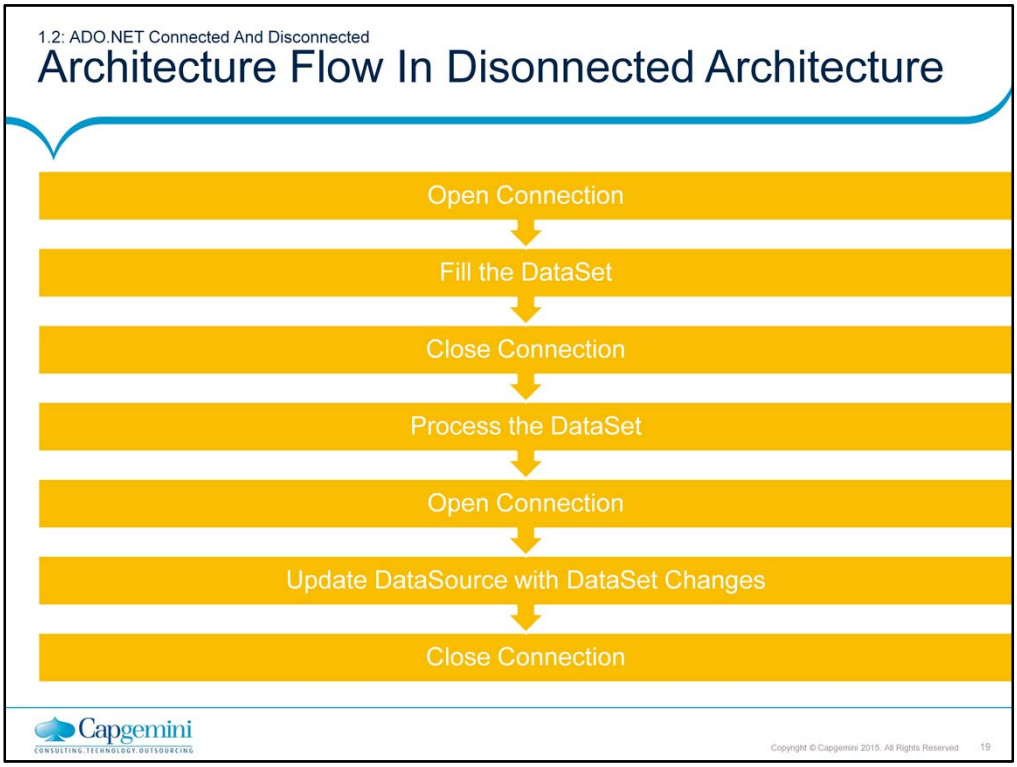
1.2: ADO.NET Connected And Disconnected

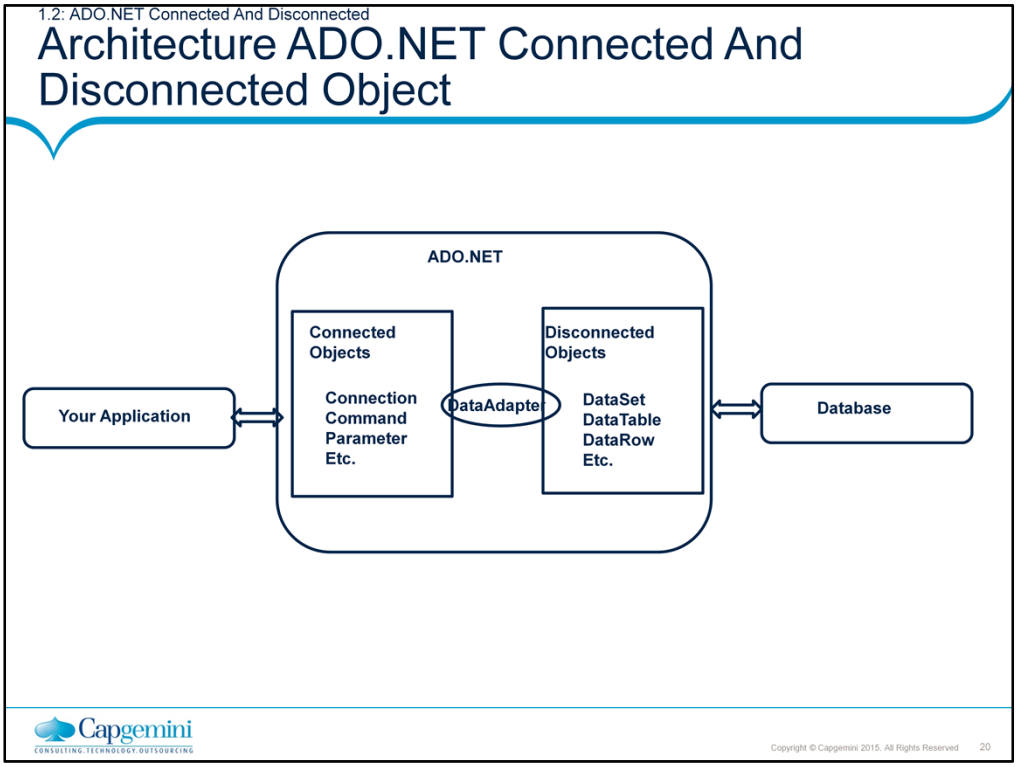
Architecture ADO.NET Disconnected Architecture

- In this environment, a subset of data from a central data store can be copied and stored in the DataSet object
- The data is modified independently & changes are merged back into the central data store
- A disconnected environment improves the scalability and performance of an applications
- Disadvantages of disconnected environment are
  - Data is not always up to date
  - Change conflicts can occur and must be resolved

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 18


1.2: ADO.NET Connected And Disconnected
Architecture Flow In Disonnected Architecture
Open Connection
Fill the DataSet
Close Connection
Process the DataSet
Open Connection
Update DataSource with DataSet Changes
Close Connection
Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING
Copyright © Capgemini 2015. All Rights Reserved
19


1.2: ADO.NET Connected And Disconnected
Architecture ADO.NET Connected And Disconnected Object
ADO.NET
Connected Objects
Connection
Command
Parameter
Etc.
DataAdapter
Disconnected Objects
DataSet
DataTable
DataRow
Etc.
Your Application
Database
Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING
Copyright © Capgemini 2015. All Rights Reserved
20

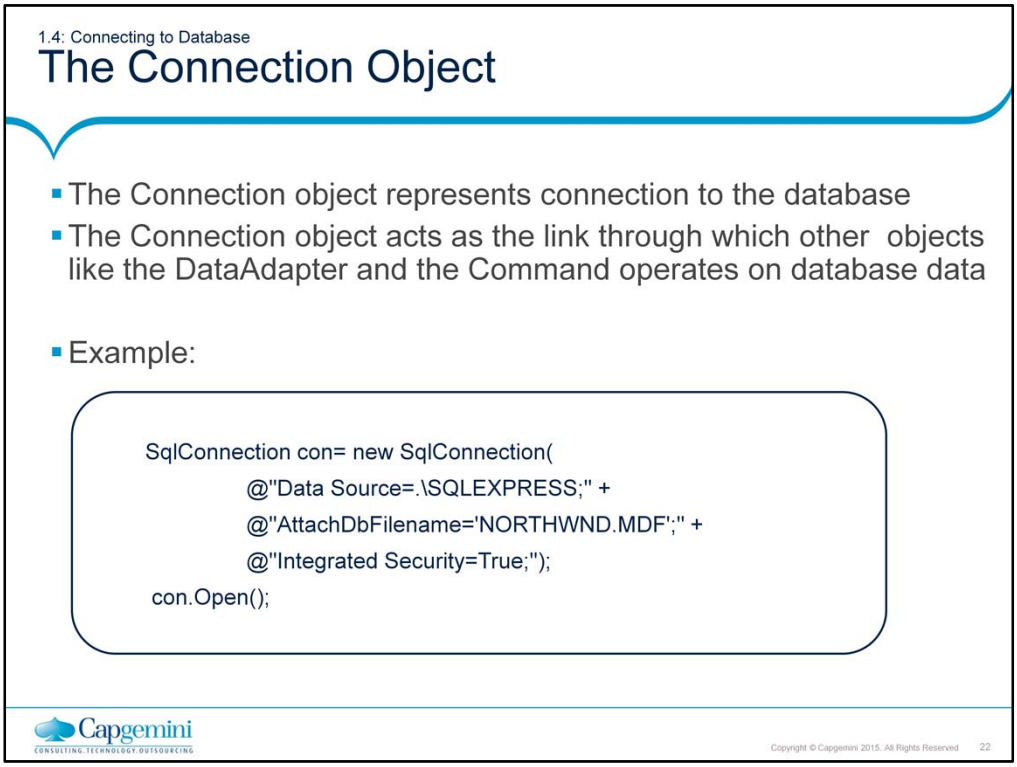
1.3: ADO.NET 4.5 Namespace Collection

ADO.NET Namespaces

System.data	Core namespace, defines types that represent data
System.Data.Common	Types shared between managed providers
System.Data.OleDb	Types that allow connection to OLE DB compliant data sources
System.Data.SqlClient	Types that are optimized to connect to Microsoft® SQL Server
System.Data.SqlTypes	Native data types in Microsoft® SQL Server

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 21



The Connection Object:

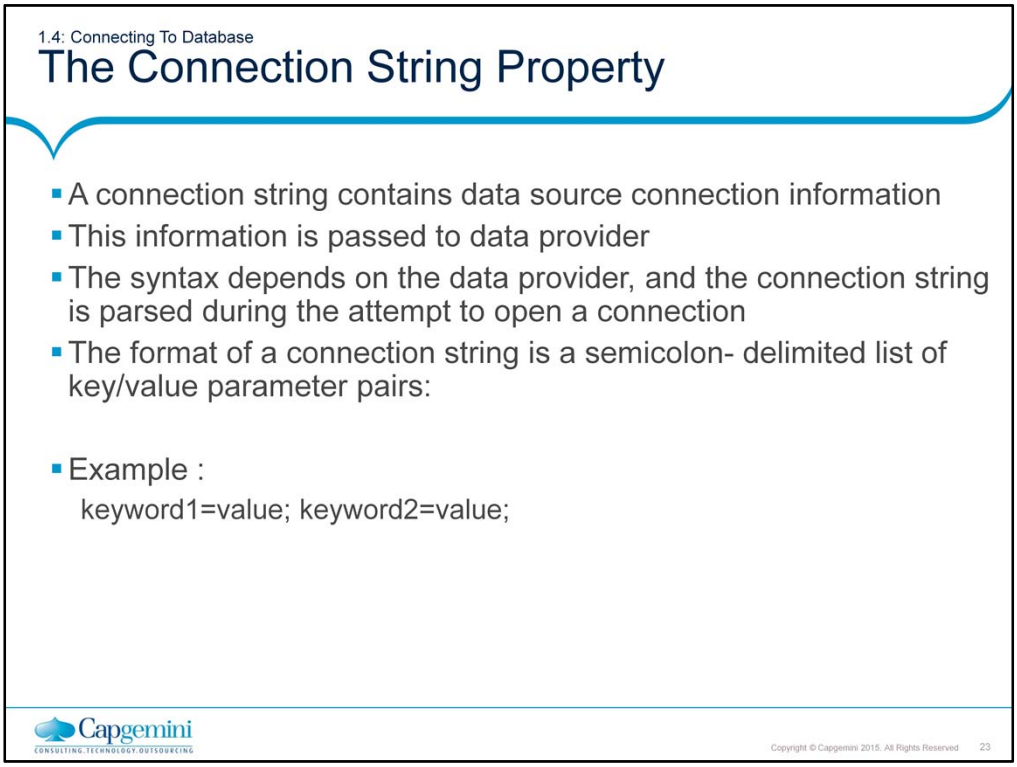
Database connections are a critical and limited resource. Connections must be managed to ensure that an application performs well and is scalable.

A connection object is used to connect to a specific data source. A SqlConnection object represents a unique session to a SQL Server data source. Similarly an OracleConnection or OLEDBConnection represents unique session to an Oracle data source or OLEDB data source respectively. SQL Server and Oracle data providers provide connection pooling, while the OLE DB and ODBC providers use the pooling provided by OLE DB or ODBC, respectively.

The SqlConnection is used together with SqlDataAdapter and SqlCommand to increase performance when connecting to a Microsoft SQL Server database. For all third-party SQL server products, and other OLE DB-supported data sources, use OleDbConnection.

When you create an instance of SqlConnection, all properties are set to their initial values. The following code examples demonstrate how to create and open connections:

For SQL Server:
Using System.Data.SQLClient;
SQLConnection con;
Con=new SQL Connection();
con.open();



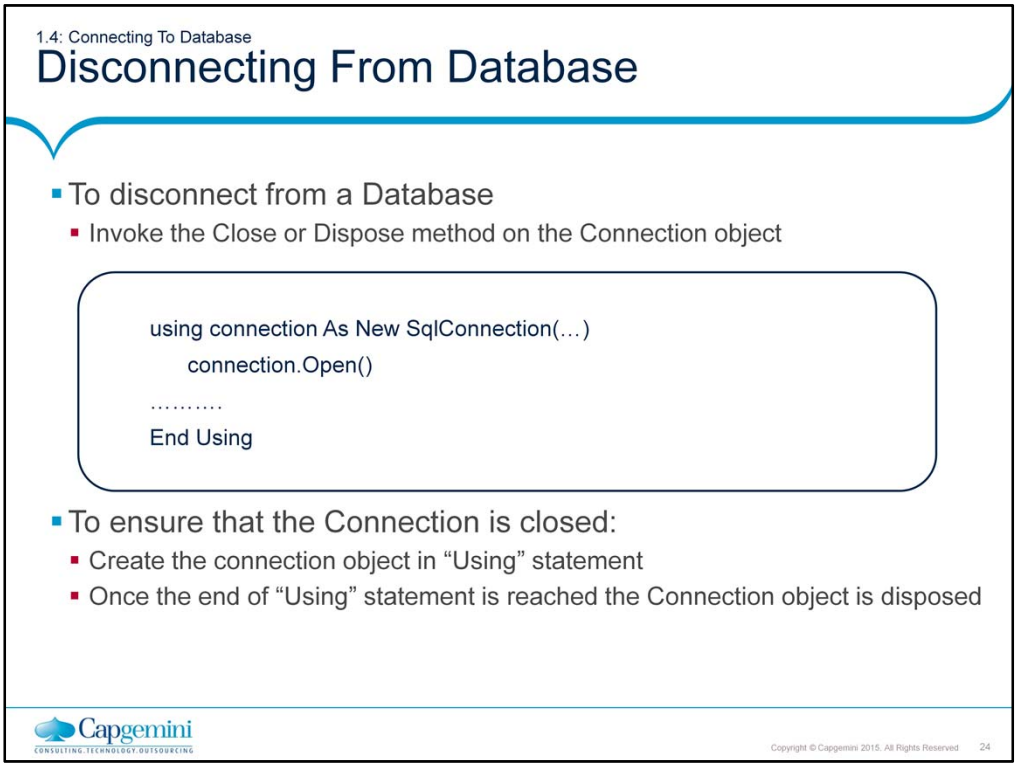
The ConnectionString Property : A connection string contains initialization information that is passed as a parameter from a data provider to a data source. The syntax depends on the data provider, and the connection string is parsed during the attempt to open a connection. Syntax errors generate a run-time exception, but other errors occur only after the data source receives connection information. Once validated, the data source applies the options specified in the connection string and opens the connection.
Keywords are not case sensitive, and spaces between key/value pairs are ignored. However, values may be case sensitive, depending on the data source. Any values containing a semicolon, single quotation marks, or double quotation marks must be enclosed in double quotation marks.

E.g.- "Data Source=SSM0107;Initial Catalog=EMS;User ID=sa;Password=sa"

Connection string parameters

Data Source - The name or network address of the instance of SQL Server to which to connect.
Initial Catalog – The name of the database.
Integrated Security – When false, User ID and Password are specified in the connection. When true, the current Windows account credentials are used for authentication.
User ID/Password – Valid User credentials to get an access to data source.
Persist Security Info - When set to false or no (strongly recommended), security-sensitive information, such as the password, is not returned as part of the connection if the connection is open or has ever been in an open state.

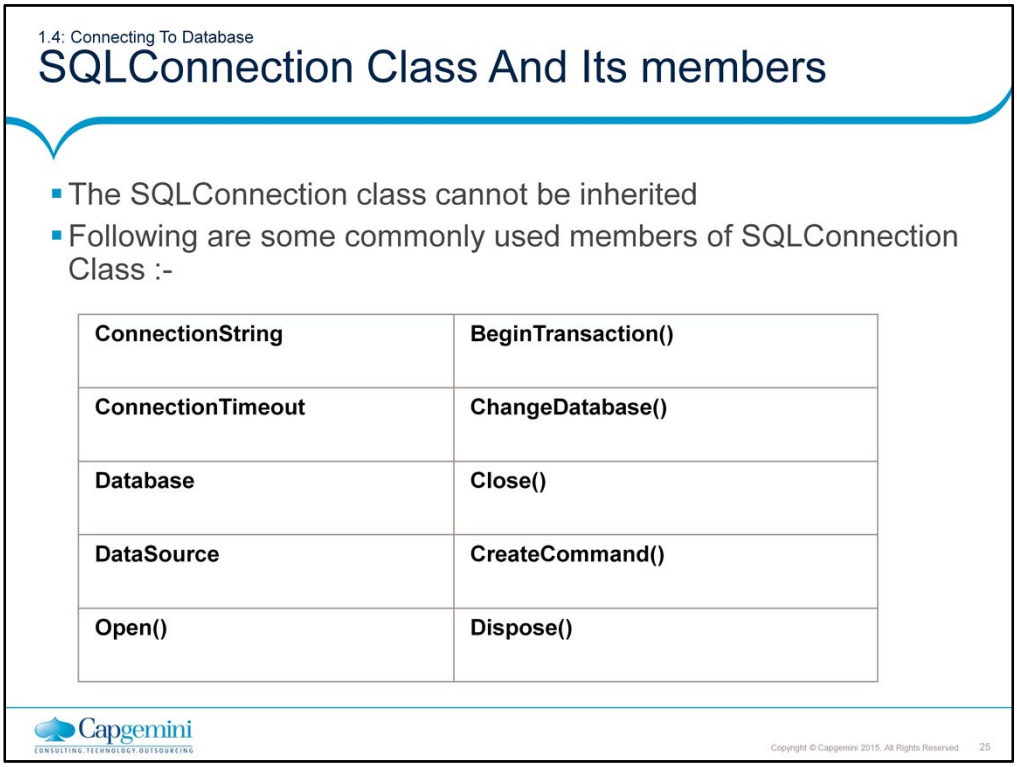
Resetting the connection string resets all connection string values including the password. Recognized values are true, false, yes, and no.



Disconnecting from Database:

Connections should be opened as late as possible and closed as soon as possible using the Close() method. Alternatively, you can create the connection in a using block to ensure that the system disposes of the connection when the code exits the block. The connection should be used as briefly as possible, meaning that connections should not last longer than a method call. Connections should not be passed between methods—in addition to creating performance problems and limiting scalability, this can lead to security vulnerabilities.

The SQLConnection object acts a link through which other objects like the DataAdaptor and the Command objects communicate with the database and submit queries and retrieve results. Whenever SqlConnection goes out of scope, it remains open. Therefore, you must explicitly close the connection by calling Close or Dispose methods. Close and Dispose are functionally equivalent.



SQLConnection class and its members:

The SQLConnection class cannot be inherited. As specified earlier, the SQLConnection represents open connection to SQL server database. The SQLConnection class exposes various properties and methods which allows a .NET programmer the ease to work with the Connection Object. Some of the members are as follows:

ConnectionString: Gets or sets the string to open a SQL Server database.
ConnectionTimeout: The wait time while trying to establish a connection before the attempt is terminated and error is generated
Database: Obtains the name of the current database or the database to be used after a connection is opened
DataSource: - Gets the name of the instance of SQL Server to which to connect
Open(): Opens a database connection with the property settings specified by the ConnectionString
BeginTransaction(): Starts a database transaction
ChangeDatabase(): Changes the current database for an open SqlConnection
Close(): Closes the connection to the database. This is the preferred method of closing any open connection
CreateCommand(): Creates and returns a SqlCommand object associated with the SqlConnection.

Working with the InfoMessage Event

You can retrieve warnings and informational messages from a SQL Server data source using the InfoMessage event of the SqlConnection object.

The following code example shows how to add an event handler for the InfoMessage event.

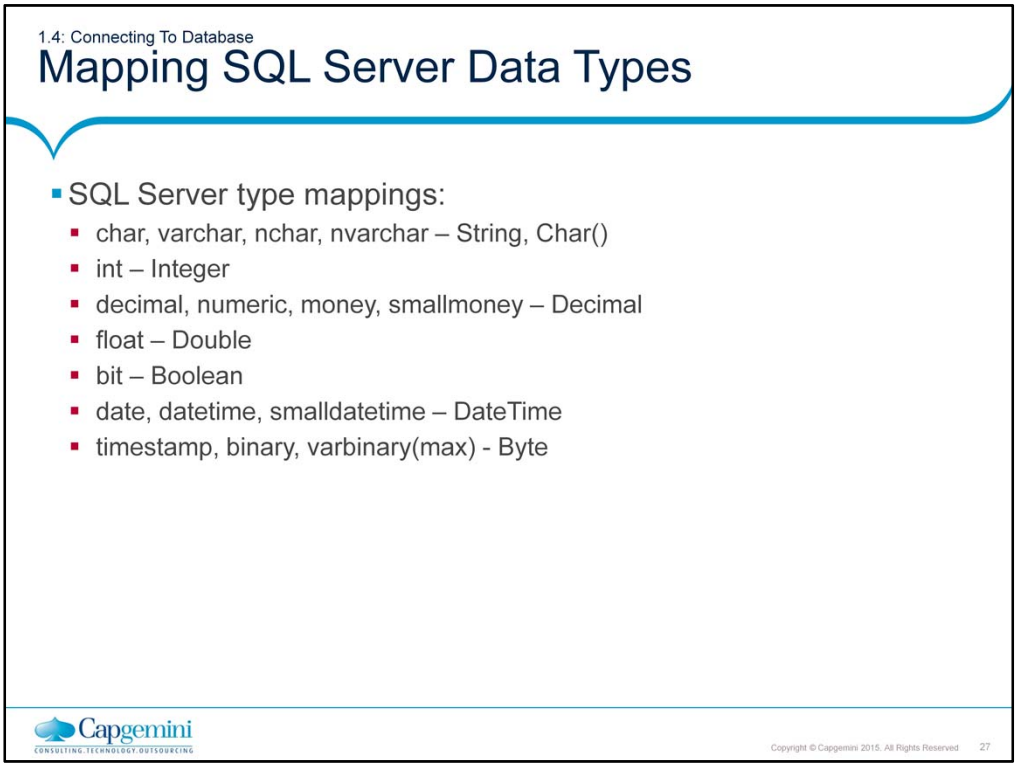
```
// Assumes that connection represents a SqlConnection object.
connection.InfoMessage += new
SqlInfoMessageEventHandler(OnInfoMessage);

protected static void OnInfoMessage(object sender,
SqlInfoMessageEventArgs args)
{
      foreach (SqlError err in args.Errors)
      {
            Console.WriteLine( "The {0} has received a severity {1}, state {2}
            error number {3}\n" + "on line {4} of procedure {5} on server
            {6}:\n{7}", err.Source, err.Class, err.State, err.Number,
            err.LineNumber,err.Procedure, err.Server, err.Message);
     }
}
```

Working with the StateChange Event

The StateChange event occurs when the state of a Connection changes. The StateChange event receives StateChangeEventArgs that enable you to determine the change in state of the Connection by using the OriginalState and CurrentState properties. The OriginalState property is a ConnectionState enumeration that indicates the state of the Connection before it changed. CurrentState is a ConnectionState enumeration that indicates the state of the Connection after it changed.

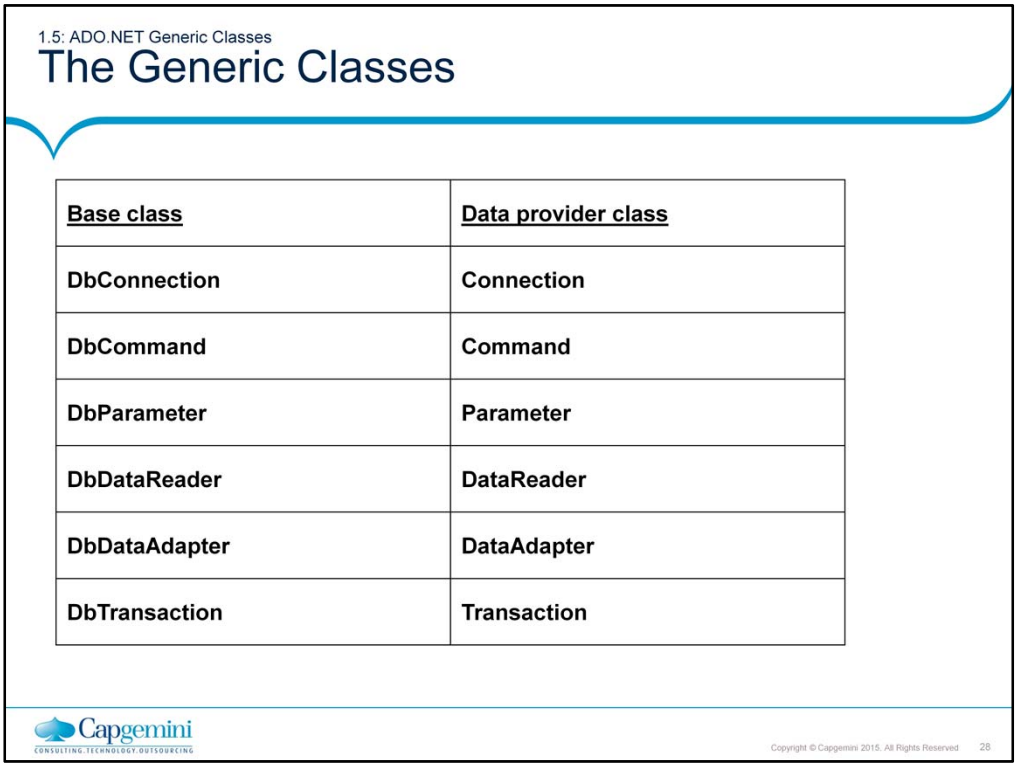
The following code example uses the StateChange event to write a message to the console when the state of the Connection changes.

```
// Assumes connection represents a SqlConnection object.
connection.StateChange += new StateChangeEventHandler(OnStateChange);

protected static void OnStateChange(object sender, StateChangeEventArgs
args)
 {
     Console.WriteLine( "The current Connection state has changed from {0} to
     {1}.", args.OriginalState,args.CurrentState);
 }
```



Mapping SQL Server Data Types to .Net Types:

Databases and .Net Framework are based on different type systems. Hence each .Net data provider defined mappings between database types and .Net Framework types to maintain data integrity when reading and writing.
In our course, since we are dealing with examples based on SQL Server we take a look at the Mapping SQL Server Datatypes to .Net Framework Datatypes.



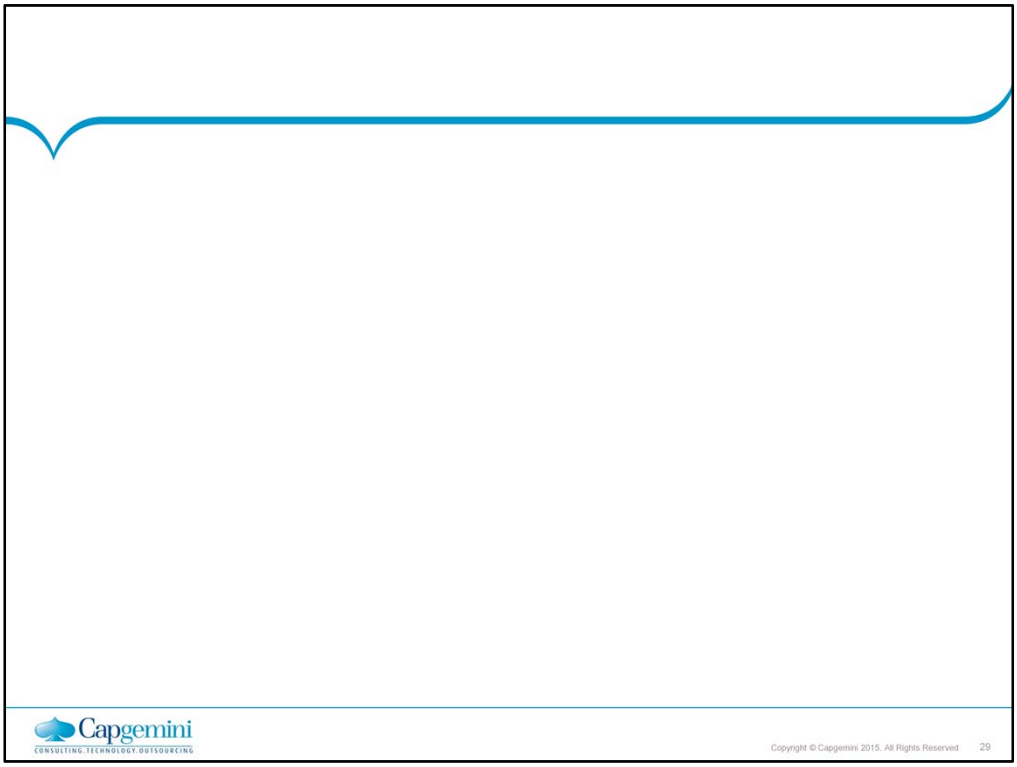
The Generic Classes:

The table represents the Generic Base Classes and their corresponding Provider Specific classes. That means SqlConnection and OracleConnection classes inherit from a common base class called DbConnection. As we progress along in the course, we will see how to use the provider specific class.

You can take advantage of polymorphism via inheritance and write generic code capable of dealing with multiple database in neutral manner. ADO.NET offers much more than common base classes. In addition to the classes mentioned in the table shown on the slide, ADO.NET also provides a set of classes called "Factory" classes. These factory classes help you to create instances of these base classes dynamically. This ability makes it possible to store the choice of your data provider in a configuration file and then at run time create instances of corresponding data provider classes. For example, SQL server data provider has a factory class called SqlClientFactory that allows you to create instances of SqlConnection, SqlCommand and so on. Similar classes exists for other data providers also.

The SqlClientFactory class has various methods shown as follows:-

CreateConnection - Creates an instance of SqlConnection class
CreateCommand - Creates an instance of SqlCommand class
CreateParameter - Creates an instance of SqlParameter class
CreateDataAdapter - Creates an instance of SqlDataAdapter class
CreateCommandBuilder - Creates an instance of SqlCommandBuilder class

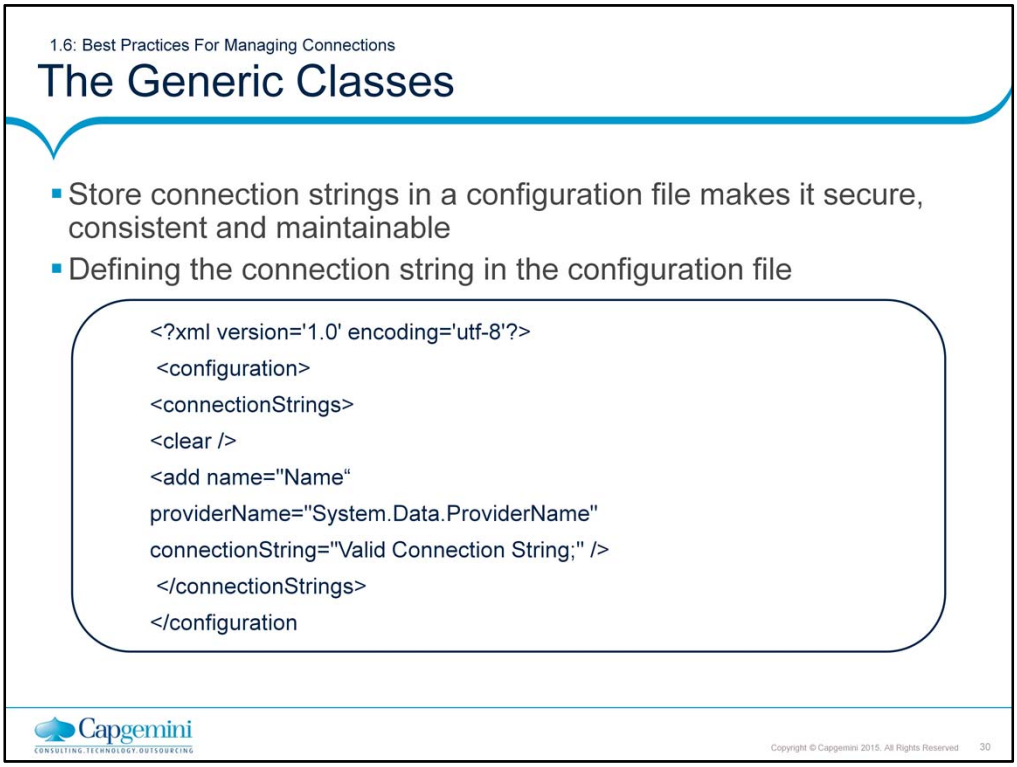


The Generic Classes:

Consider the following code snippet:

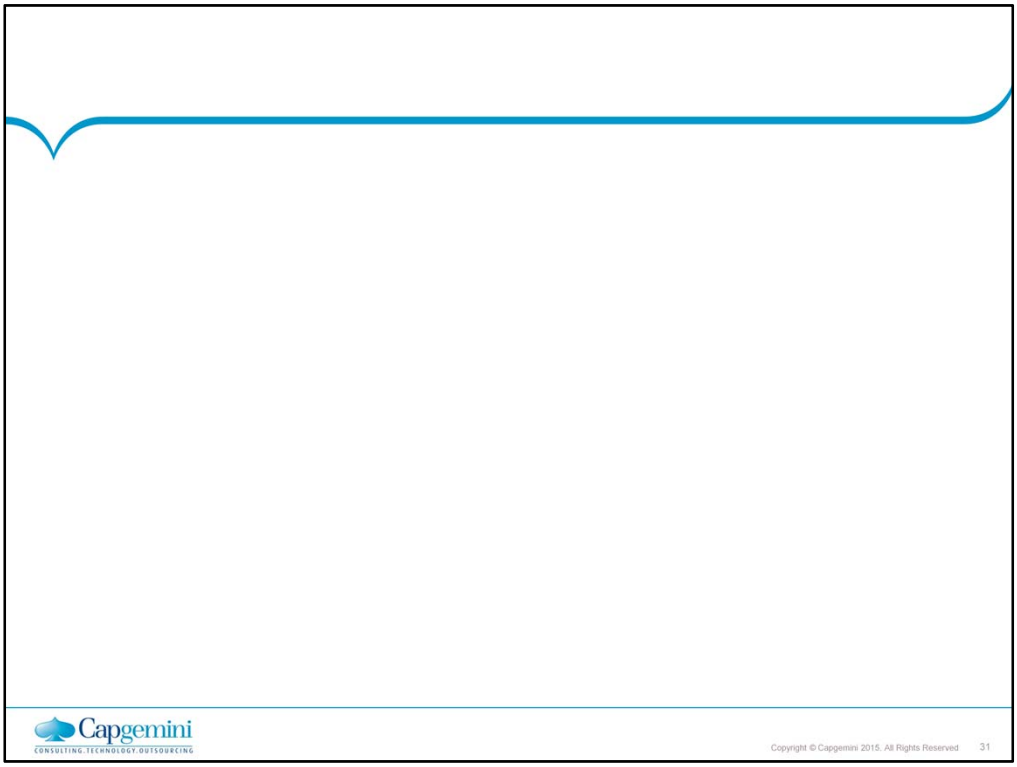
```
public void ExecuteQuery(string sql,string provider)
{
   DbConnection cnn=null;
   DbCommand cmd=null;
   DbProviderFactory factory = null;
         switch(provider)
          {
            case "sqlclient":
            factory = SqlClientFactory.Instance;
            break;
            case "oracleclient":
            factory = OracleClientFactory.Instance;
            break;
         }
    cnn = factory.CreateConnection();
    cmd = factory.CreateCommand();
   //now use cnn and cmd as usual to execute a query
}
```

Note, the usage of Instance property of SqlClientFactory and OracleClientFactory classes to get an instance of corresponding factory class.

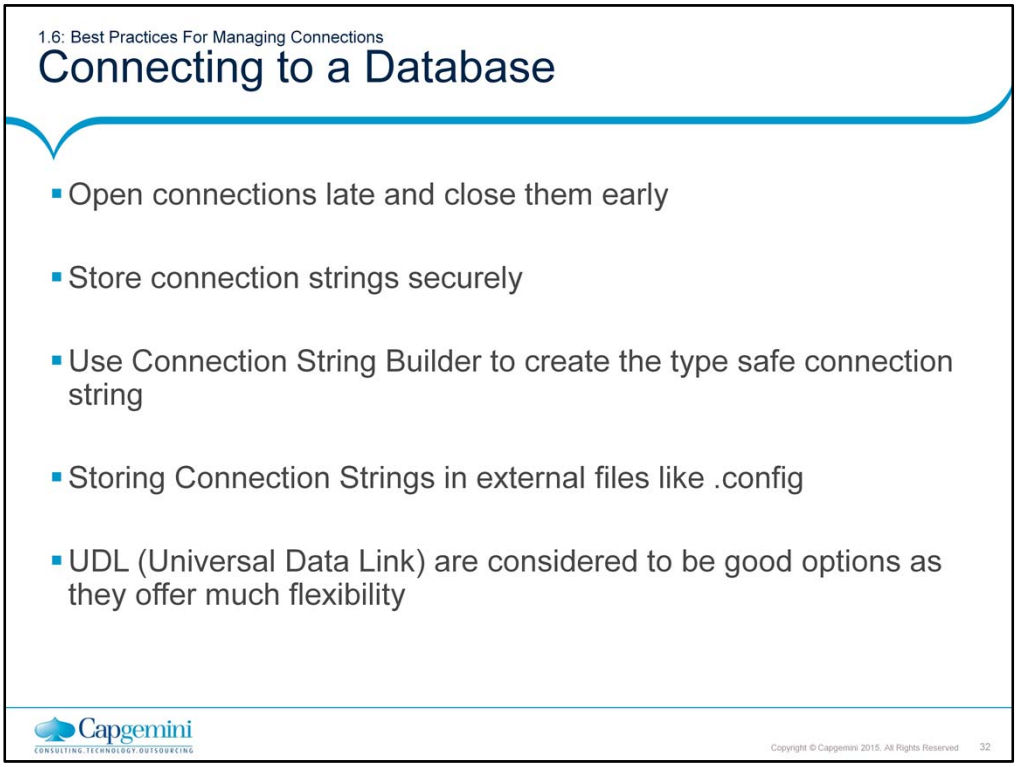


Storing Connections in Configuration file:

Database connections are a critical and limited resource. Connections must be managed to ensure that an application not only performs well but is also scalable. Data Providers use a connection string to establish connection to the database. Where should a developer store the connections string? Ideally the connection string should be stored in such a place wherein it increases maintainability of the application and also eliminates the need to recompile the application when it is modified.
The popular option would be store the connection string in the configuration file. An application configuration file is an XML based file to store application specific settings. Storing the connection string in the configuration file makes it consistent, secure and maintainable.
The connectionStrings Section - Connection strings can be stored as key/value pairs in the connectionStrings section of the configuration element of an application configuration file. Child elements include add, clear, and remove. The connection string can be placed in the configuration file as shown on the slide.




```
// Retrieves a connection string by name.
// Returns null if the name is not found.
static string GetConnectionStringByName(string name)
{
// Assume failure.
string returnValue = null;
// Look for the name in the connectionStrings section. ConnectionStringSettings
settings = ConfigurationManager.ConnectionStrings[name];
 // If found, return the connection string.
 if (settings != null)
returnValue = settings.ConnectionString;
return returnValue;
}
```



Managing Connections:
Database connections represents a very critical, expensive, and limited resource. On optimizing how you use them is therefore very important for any application; not just for .NET Framework applications, but in particular for multi-tier Web applications. The bottom line for database connections can be summarized in the following two points:

Store connection strings securely
Open connections late and close them early

For connecting to a database, you can follow the steps mentioned below:

Define a string and assign the connection information to it.
Create the connection object using the connection string.
Invoke the open method on the Connection object.
Close the Connection object when not in use.

For creating a type safe connection you can use the ConnectionBuilder object as follows:

```
SqlConnectionStringBuilder sb= New
SqlConnectionStringBuilder()
sb("Data Source") = “192.166.67.176”
sb("Initial Catalog") = “Northwind"
String  conn=sb.ConnectionString;
```

Keep connections open to the database only when required. Reduce the number of times you open and close a connection for multiple operations.

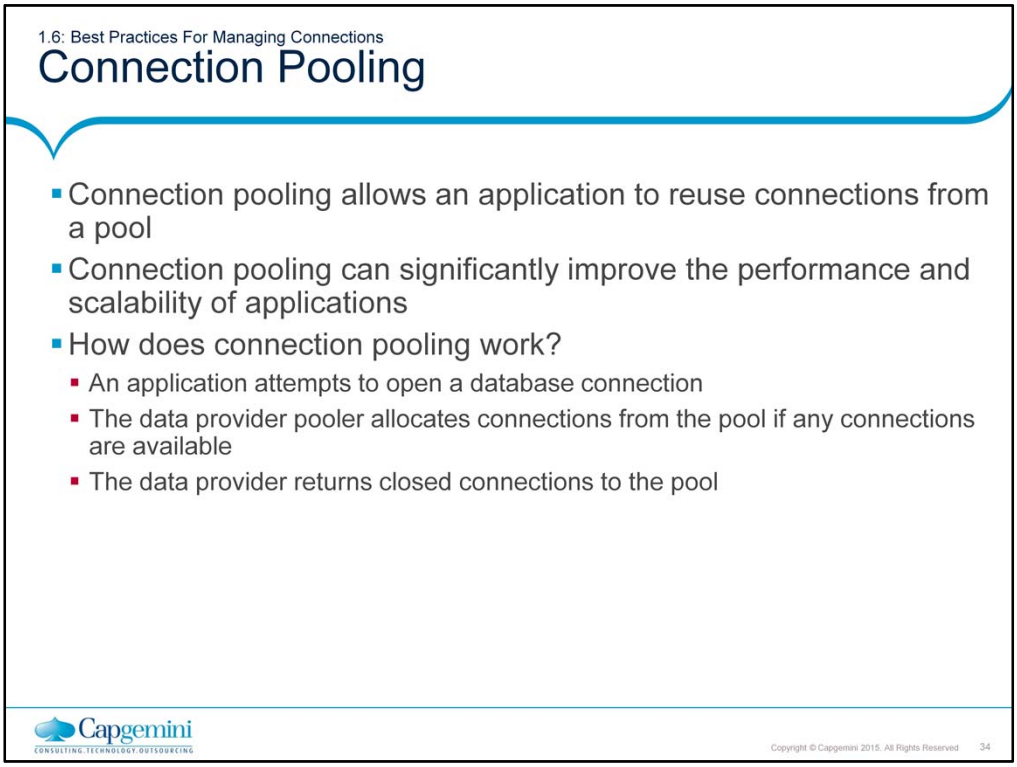
1.6: Best Practices For Managing Connections

Connecting to a Database

- Every technique of storing connection string suffers from a security threat
- We can have a tailor-made engine for encrypting and decrypting connection details

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

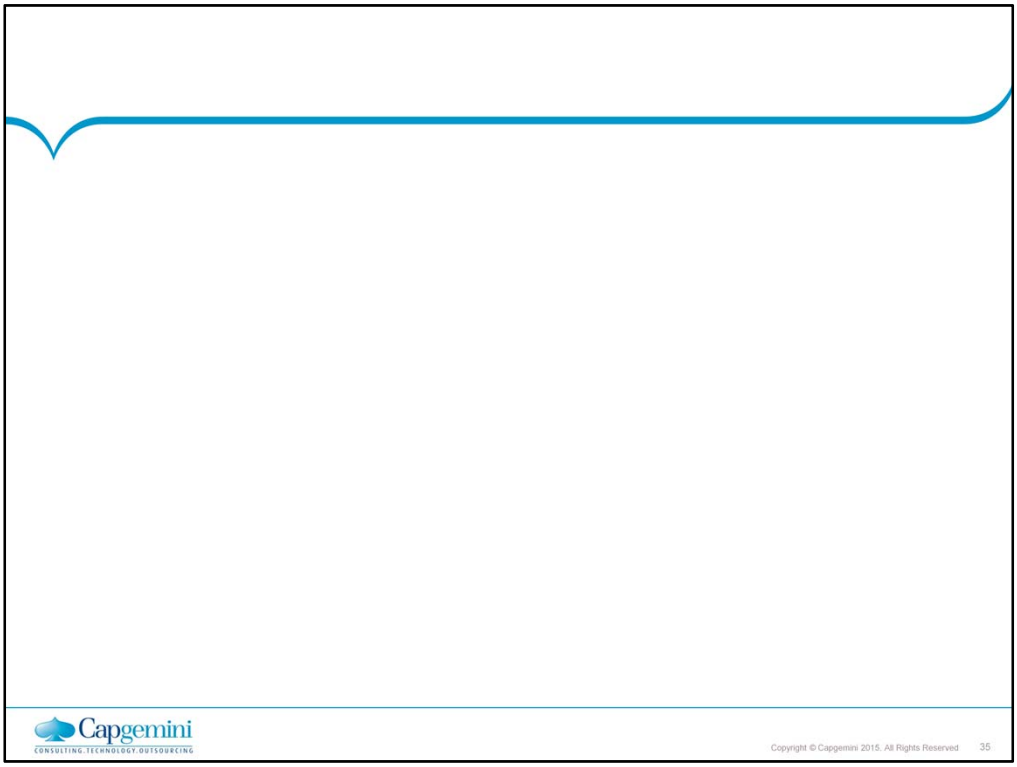
 33



Connection Pooling:

Connection pooling enables connections to be reused efficiently instead of repeatedly creating and destroying new connections. Data Providers pool connections that have same security context. SQL Server and Oracle data providers provide connection pooling. If Integrated Security is used then Connection Pool is created for each user accessing the client system, and if userid and password is used then single connection pool is maintained across for the application. In case of using user id and password, each user can use the connections of the pool created and then released to the pool by other users. This option is recommended for better performance experience for the end user.

A connection pool is created for each unique connection string. An algorithm associates items in the pool based on an exact match with the connection string; this includes capitalization, order of name/value pairs, and even spaces between name/ value pairs.



Properties used while specifying connection string to implement Connection Pooling .

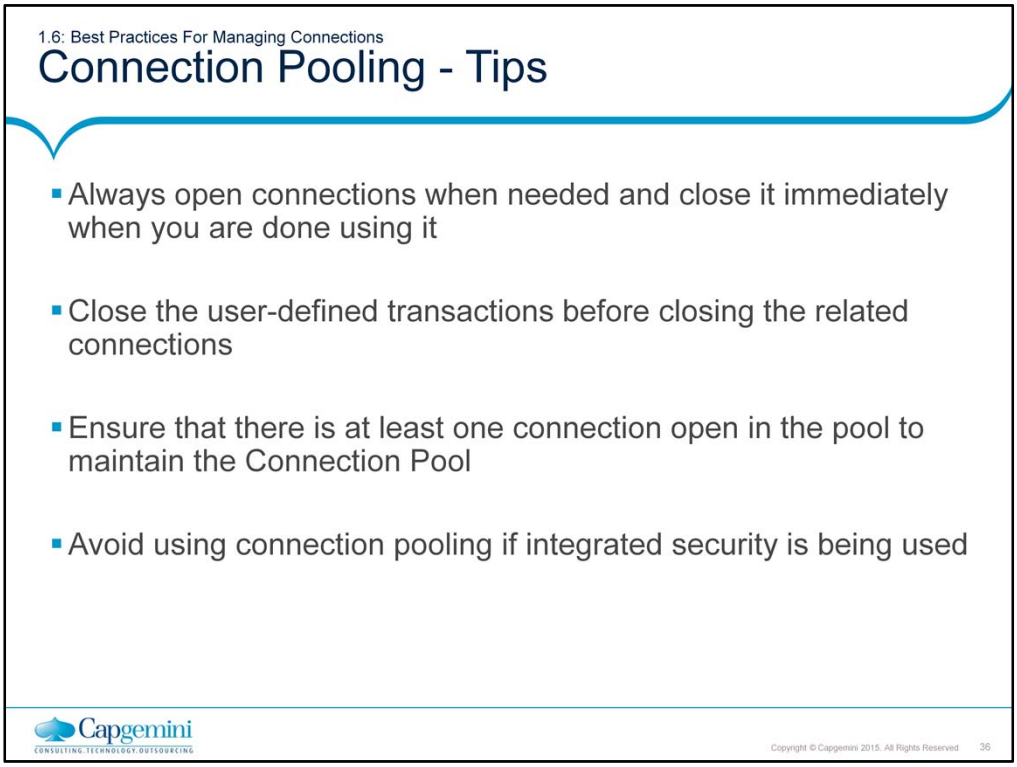
Connection Timeout – Default is 15. Maximum Time (in secs) to wait for a free connection from the pool.

Min Pool Size – Default is 0. The minimum number of connections allowed in the pool.

Max Pool Size – Default is 100. The maximum number of connections allowed in the pool.

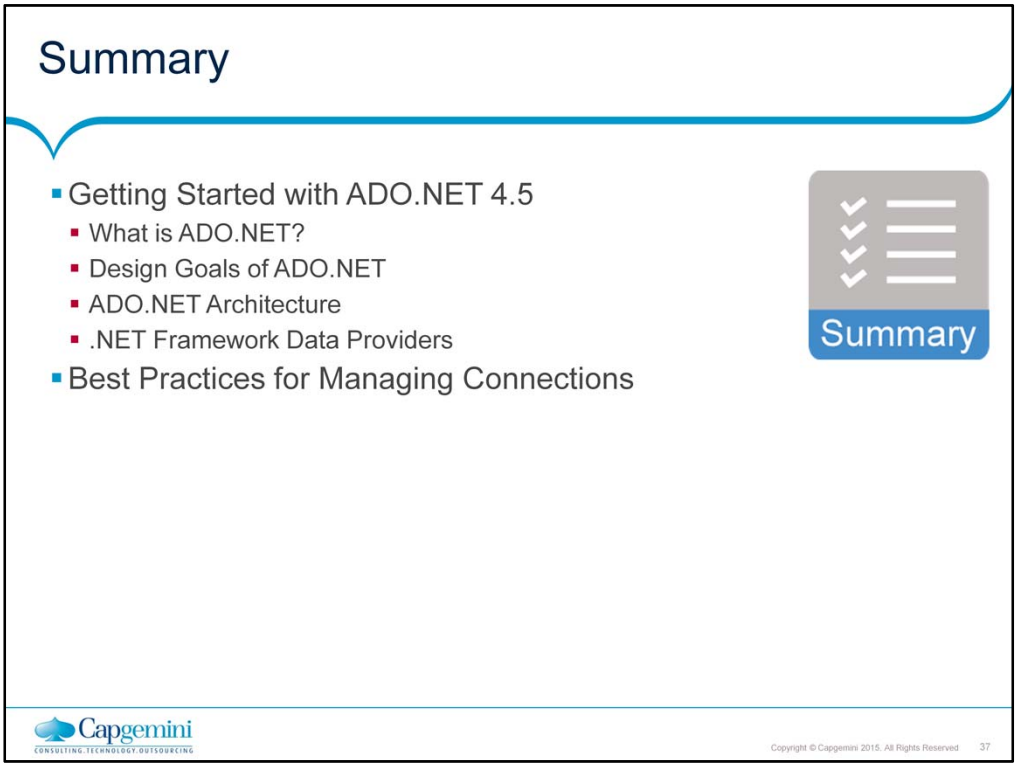
Incr Pool Size – Default is 5. Controls the number of connections that are established when all the connections are used.

Decr Pool Size – Default is 1. Controls the number of connections that are closed when an excessive amount of established connections are unused.



Connection Pooling – Tips

Only open connections when needed. That is, timing is everything, so open a connection just before you need it and not any sooner. Also, close that connection as soon as you are finished with it—don't wait for the garbage collector to do it.
Close user-defined transactions before closing related connections.
To maintain the connection pool, you should keep at least one connection open. Therefore, do not close all your connections in the pool. If server resources become a problem, you may close all connections, and the pool will be recreated with the next request.
Do not use connection pooling if integrated security is utilized. This results in a unique connection string per user, so each user has a connection pool that is not available to other users. The end result is poor performance, so pooling should be avoided in this scenario.



Add the notes here.

Review Question

- Question 1: Which of the following are design goals of ADO.NET?
  - Using current knowledge of ADO
  - Support for N-Tier programming
  - None of the above

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 38

Add the notes here.

Review Question: Match the Following

1. Numeric	DateTime
2. Float	Double
3. Bit	Decimal
4. Date	Byte
5. TimeStamp	Boolean

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 39

Add the notes here.

ADO.NET 4.5

Lesson 2: Working with Connected Architecture

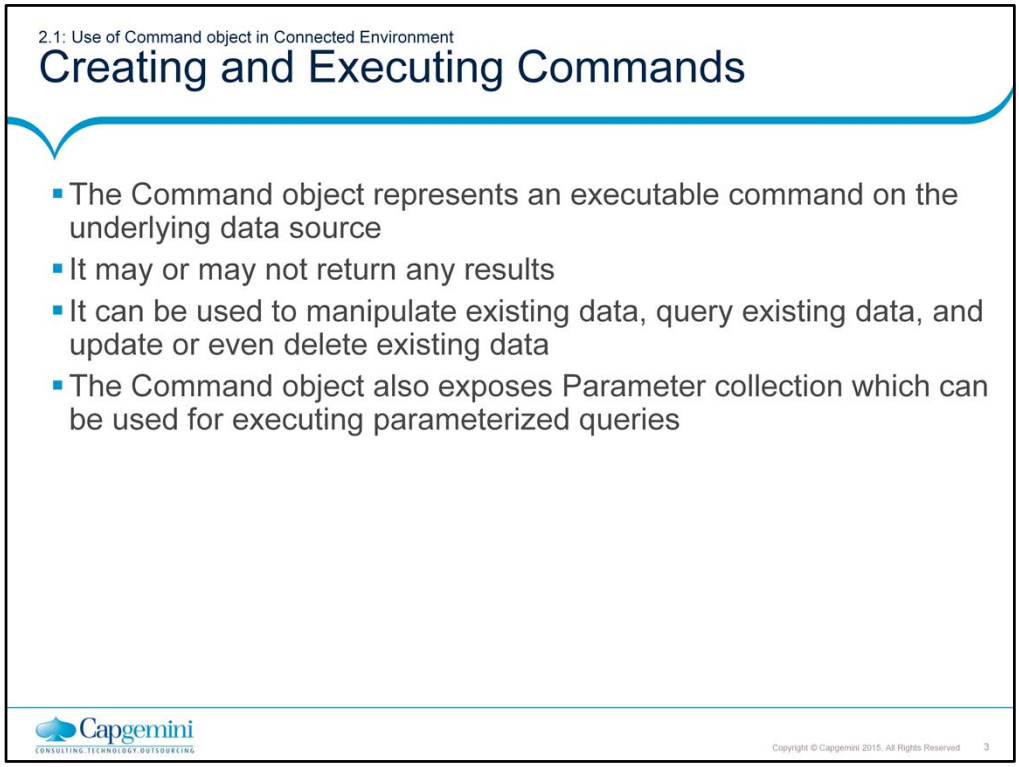
Lesson Objectives

- In this lesson, you will learn:
  - Use of SqlCommand to query and modify data
  - Use of commands to manipulate data

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 2

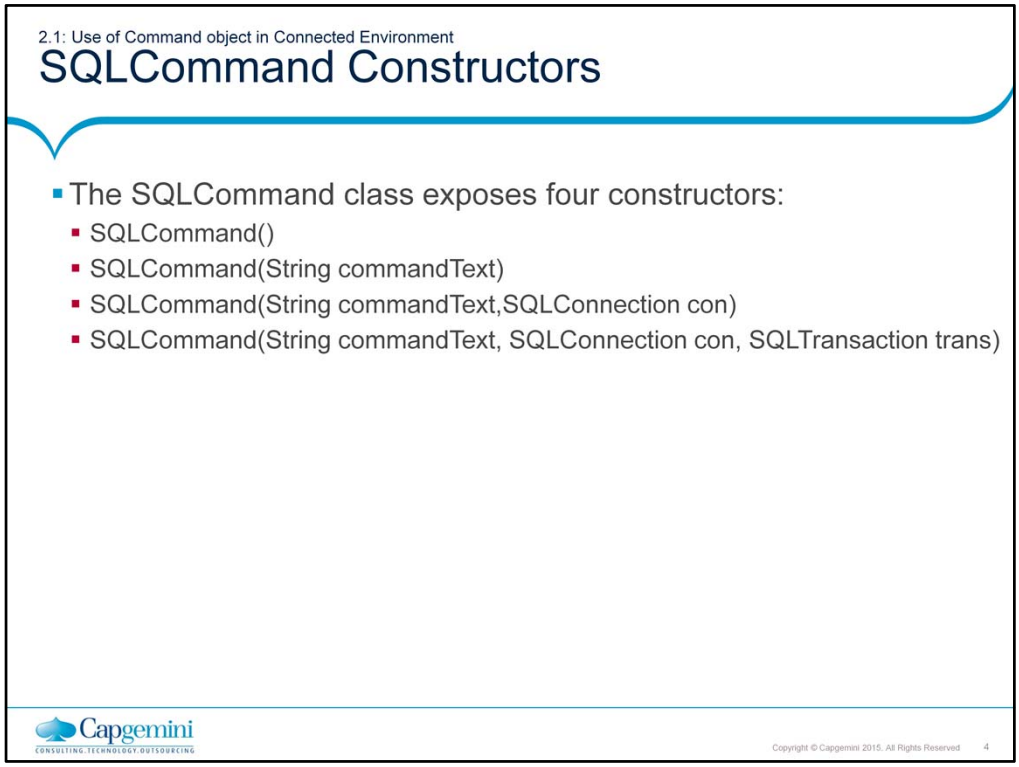
In the previous lesson, we have learnt how to connect to the database and also got familiar to objects provided by .Net Provider. In this lesson we will learn to retrieve and modify data using the Command object and the related objects.



Creating and Executing Commands:

The command classes (OracleCommand and SqlCommand) are used to execute commands on a data source across a connection. Once a connection is established, a command can be used to execute a SQL statement or stored procedure. The SQLCommand represents a TransactSQL statement or stored procedure to execute against a SQL Server database. The SQLCommand class is part of the System.Data.SQLClient namespace. When an instance of SqlCommand is created, the read/write properties are set to their initial values.

A Command object executes SQL statements and stored procedures against the data source using an established Connection. The CommandText property of the Command class contains the SQL statement executed against the data source. The Command object is specific to the type of data source—for example, the .NET Framework data provider for SQL Server includes the SqlCommand object.



SQLCommand Constructors:

The SQLCommand class exposes four constructors:

```
SQLCommand cmd=new SQLCommand();
```

Creates a new SqlCommand with no properties set and no associated connection.

```
SQLCommand cmd = new SqlCommand(String CommandText);
```

Creates a new SqlCommand with the CommandText property set but no associated connection.

```
SQLCommand cmd = new SqlCommand(String CommandText,
SqlConnection con);
```

Creates a new SqlCommand with the CommandText property set and the specified SqlConnection as the Connection property.

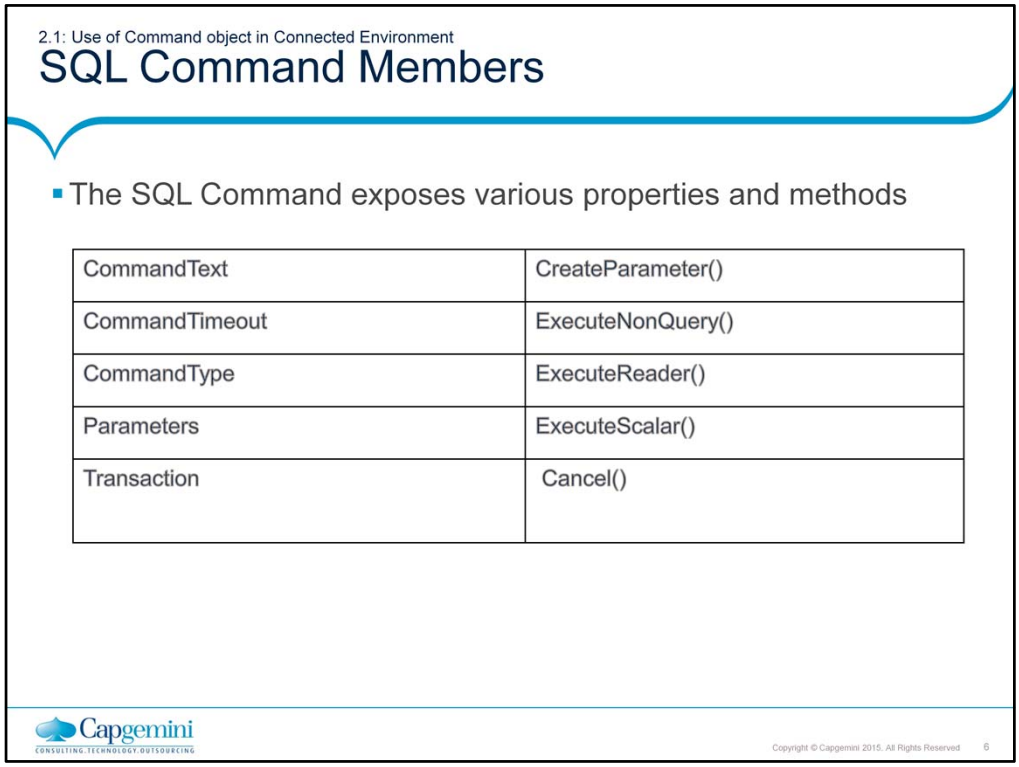
SQLCommand Constructors (Contd.):

```
SQLCommand cmd = new SqlCommand(String CommandText, SqlConnection con,
SqlTransaction trans);
```

Creates a new SqlCommand with the CommandText property set, the specified SqlConnection as the Connection property and SQLTransaction to for the command to execute in.

The following code snippet shows the creation of a command object.

```
//We assume that the SqlConnection object with a name con has already
//been created and initialized
string queryString = "SELECT OrderID, CustomerID FROM Orders;";
SQLCommand cmd=new SQLCommand(queryString,con);
con.open();
```



SQLCommand Members:

The SQLCommand class also exposes various properties and methods. Some of them are as follows:

CommandText - Gets or sets the Transact-SQL statement or stored procedure to execute at the data source.

CommandTimeout- Gets or sets the wait time before terminating the attempt to execute a command and generating an error.

CommandType- Gets or sets a value indicating how the CommandText property is to be interpreted either Text or TableDirect or StoredProcedure. Wherein the command type as text indicates a SQL statement, TableDirect indicates a table name whose columns are returned and StoredProcedure specifies the name of the stored procedure.

Parameters -Retrieves the SqlParameterCollection.

Transaction- Specifies the transaction in which the SqlCommand executes.

CreateParameter()- This method creates a new instance of a SqlParameter object

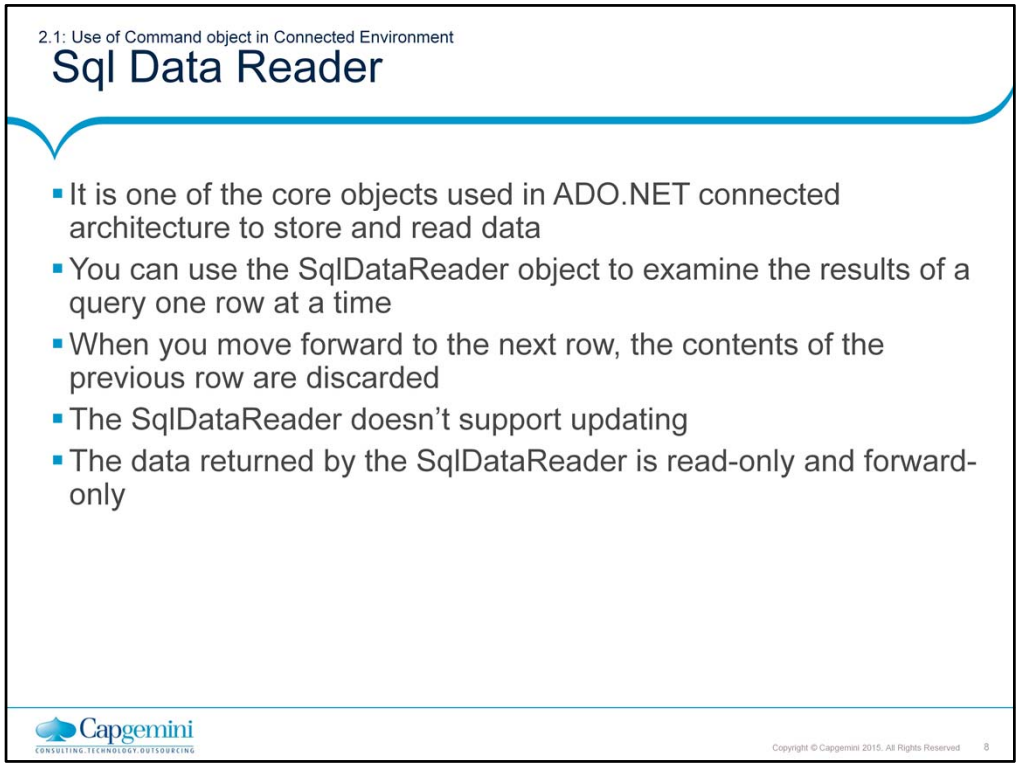
ExecutingNonQuery() - This method is used for executing commands that do not return result set, such as update, insert or delete commands. The ExecuteNonQuery method return an integer value indicating the number of rows affected by the command. For all other types of commands –1 is returned.

SQLCommand Members:

ExecuteReader()-This method sends the CommandText to the Connection and builds a SqlDataReader.

ExecuteScalar()- The ExecuteScalar() method returns the first column of the first row in the result set; all other columns and rows are ignored. This method is particularly useful for returning aggregate values, such as the result of a "SELECT COUNT(*)" SQL statement. Using this method is less code intensive, and requires fewer system resources than using ExecuteReader() and then invoking the DataReader.Read() method to get the returned value. The ExecuteReader() method creates a datareader stream on the connection, preventing anything from using the connection until the datareader is closed; ExcecuteScalar() does not do this.

Cancel() – Tries to cancel execution of a SqlCommand.

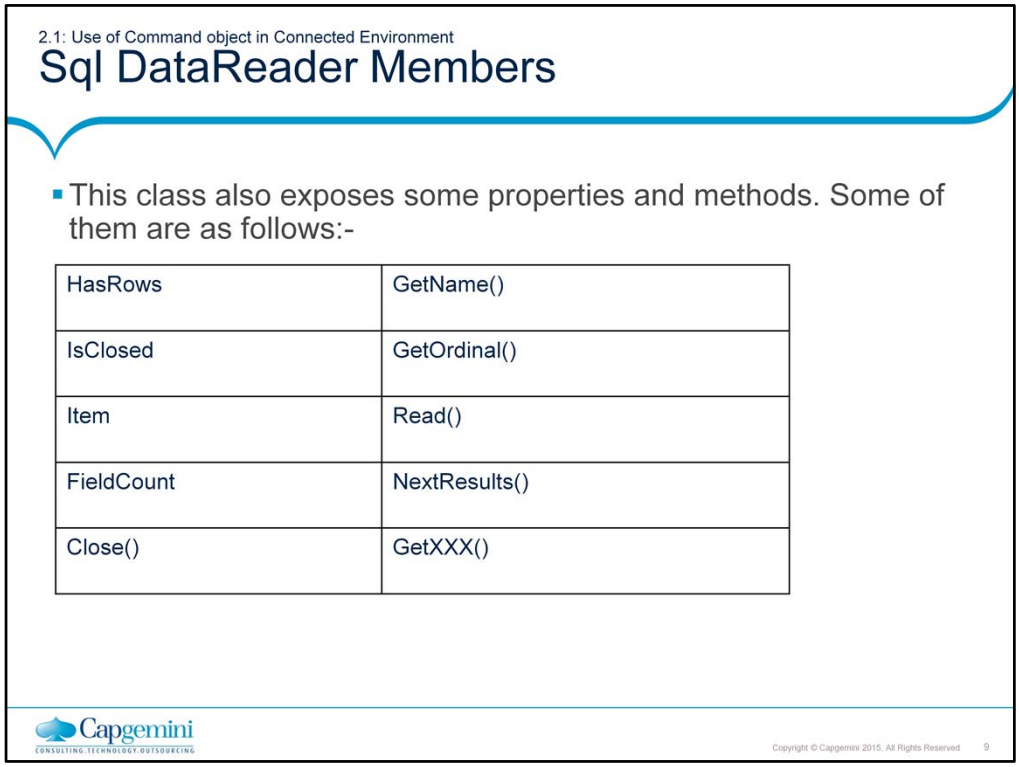


SqlDataReader:

SqlDataReader provides a means of reading a forward-only stream of data records from a SQL Server data source. To create a SqlDataReader, you must call the ExecuteReader method of the SqlCommand object, instead of directly using a constructor.

```
SqlDataReader reader = command.ExecuteReader();
```

Because the SqlDataReader object supports such a minimal set of features, it's extremely fast and lightweight. The disadvantage of using a SqlDataReader object is that it requires an open database connection and increases network activity. The SqlDataReader is a good choice when retrieving large amounts of data; only one row of data will be cached in memory at a time. You should always call the Close method when you are through; using the SqlDataReader object, as well as closing the SqlDataReader object's database connection. Otherwise the connection won't be closed until the Garbage Collector gets around to collecting the object. One possible way is to use the CloseConnection enumeration on the ExecuteReader method. This tells the Command object to automatically close the database connection when the SqlDataReader's Close method is called.



SqlDataReader Members:

The SqlDataReader class also has various properties and methods. These are some of the members:-

HasRows - Gets a value that indicates whether the SqlDataReader contains one or more rows

IsClosed - Indicates whether the data reader is closed by retrieving a value

Item – Gets the value of a column in its native format

FieldCount - Retrieves number of columns in the current row

Close()- Closes the SqlDataReader object

GetName()- Gets the name of the specified column

GetOrdinal()- Gets the column ordinal, by providing the name of the column

Read()- Moves forward the SqlDataReader to the next record

NextResults() - Moves forward the data reader to the next result, when reading the results of batch Transact-SQL statements.

GetXXX() - Gets the value of the specified column as a XXXX where XXXX is a datatype. For example, GetBoolean(), GetChar(), GetDecimal()

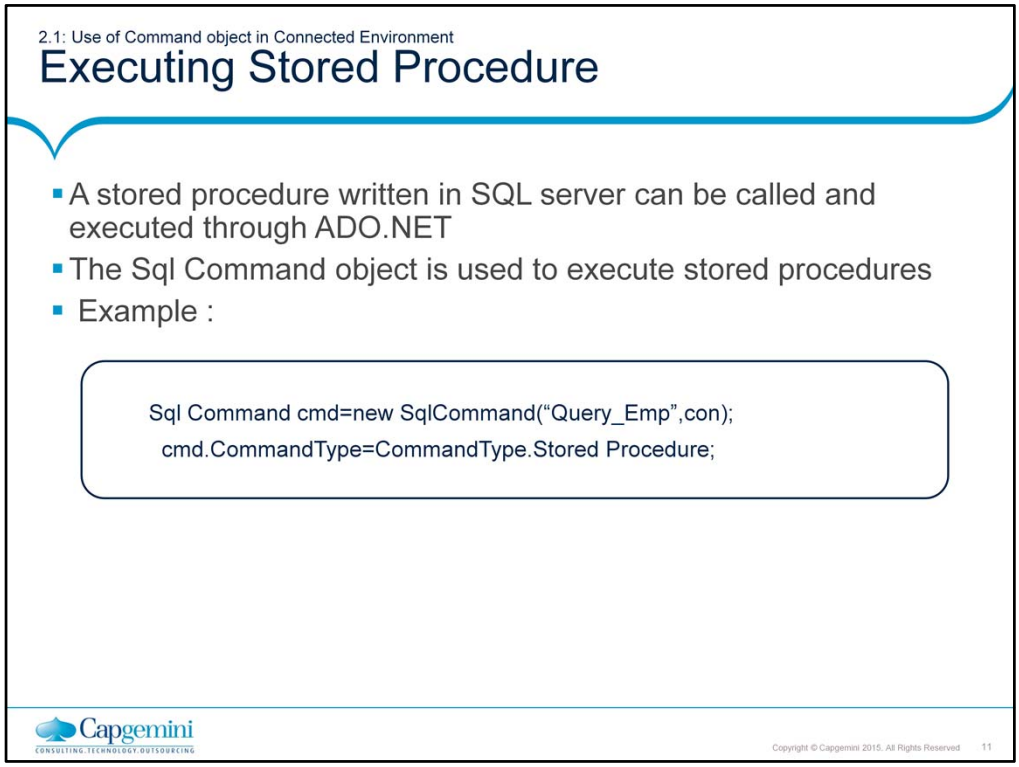
2.1: Use of Command object in Connected Environment

Demo

- Using SQL Command
- Executing Query returning a single value
- Executing Query returning a Result Set and using the Data Reader object

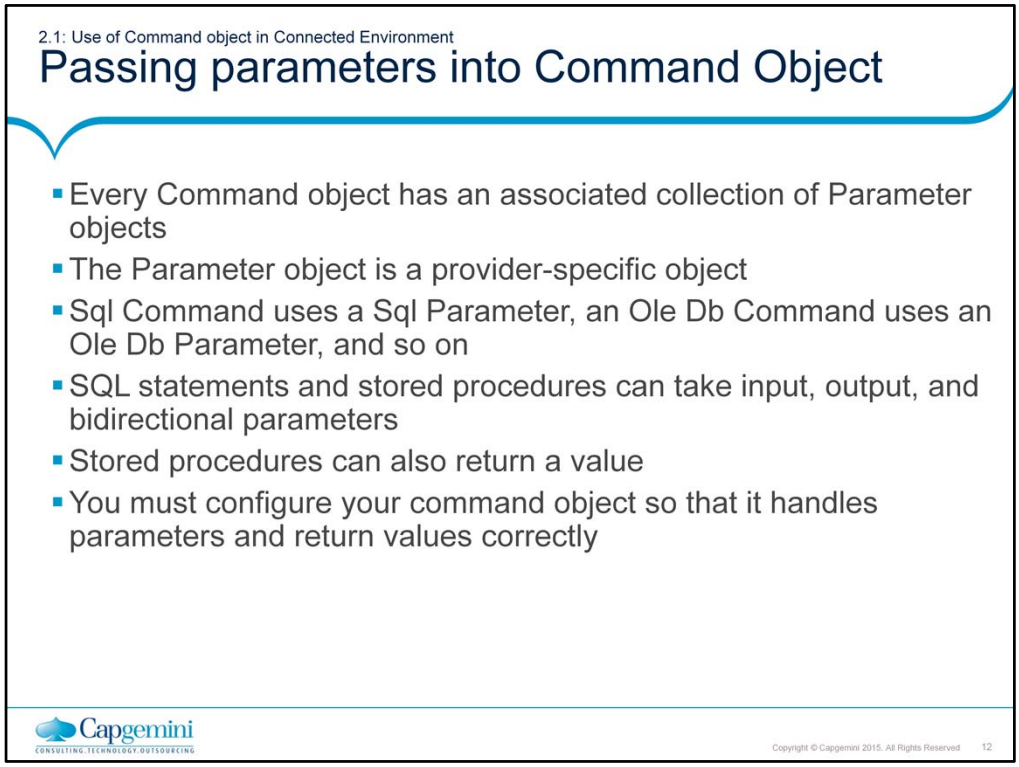
Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 10

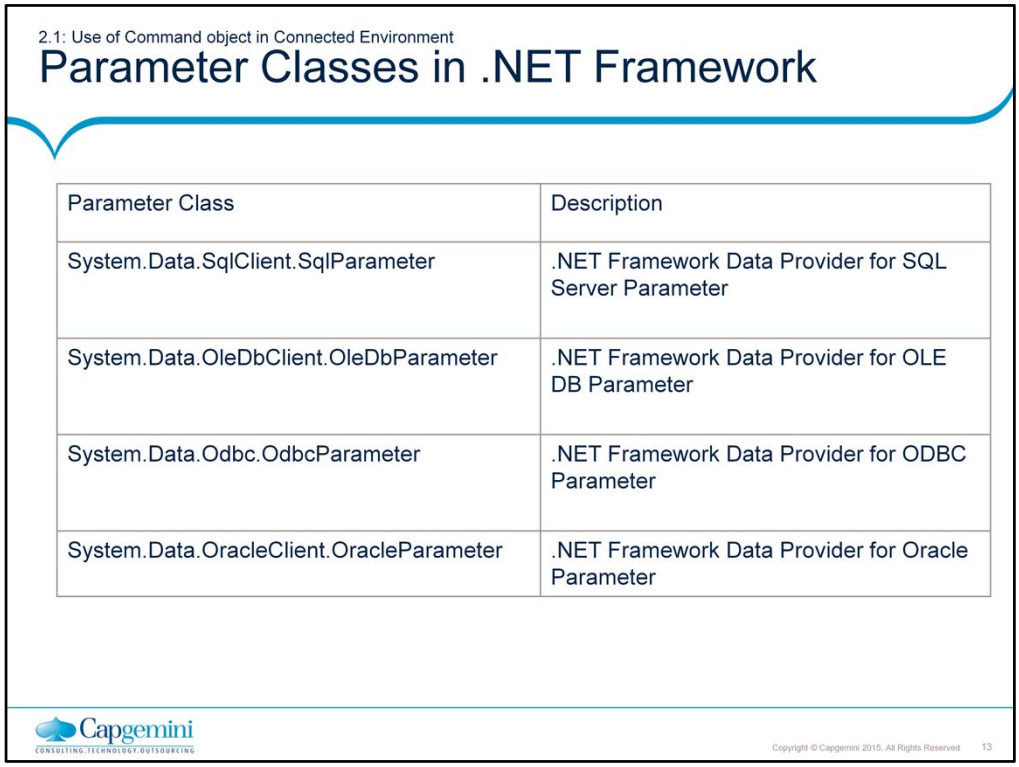


Executing Stored Procedure:

A stored procedure is a reusable sub routine stored in a database. SQL Server compiles stored procedures which makes it more efficient to use. Therefore , rather than dynamically building queries in the code, we can always take advantage of the reusability and performance benefits of stored procedures.
A stored procedure can be called by simply passing the stored procedure name followed by parameter arguments as an SQL statement. But you can use the Parameters collection of the ADO.NET Command object which enables you to more explicitly define stored procedure parameters as well as to access output parameters and return values. To call a stored procedure, set the CommandType of the Command object to StoredProcedure.



Once the CommandType is set to StoredProcedure, you can use the Parameters collection to define parameters. A Parameter object can be created using the Parameter constructor, or by calling the Add method of the Parameters collection of a Command. Parameters.Add will take as input either constructor arguments or an existing Parameter object. When setting the Value of a Parameter to a null reference, use DBNull.Value.
For parameters other than Input parameters, you must set the ParameterDirection property to specify whether the parameter type is InputOutput, Output, or ReturnValue. The following example shows the difference between creating Input, Output, and ReturnValue parameters.



Creating and Initializing Parameter Objects

Following are some of the most commonly used properties that you can use while creating and initializing parameter object.

ParameterName – Set this property to the name of the Parameterin the SQL Statement or Stored Procedure.

DbType – Set this property to the data type of the parameter.

Size – Set this property to indicate the size of the parameter, for example number of characters in string parameter. It is not necessary to specify size for data types of known and fixed sizes such as DbType.Int32.

Direction – Set this property to indicate whether, the parameter is an input parameter, output parameter or bidirectional parameter or Stored Procedure return Value. We cab use one of the values specified by ParameterDirection Enumeration.

For example, ParameterDirection.Input, ParameterDirection.Output, ParameterDirection.InputOutput or ParameterDirection.ReturnValue.

5. Value - For input or bidirectional parameters, set the Value property before you run the command. For output, bidirectional parameters and for stored procedure return value, you can retrieve the Value property after you run the command.

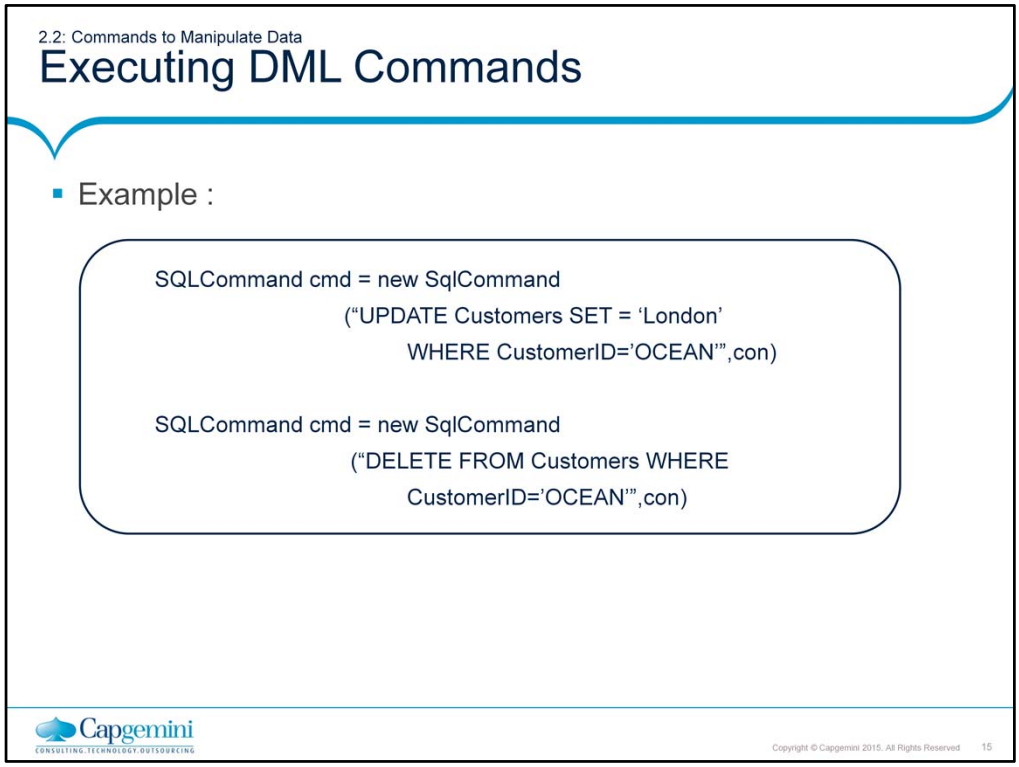
2.1: Use of Command object in connected environment

Demo

- Using SQL Command
  - Executing Stored procedure

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 14



As previously mentioned to execute DML Commands using the SQLCommand object, users can use the ExecuteNonQuery() method.
The slide shows some code snippets of how DML commands can be executed. You could also pass parameters to the DML commands.

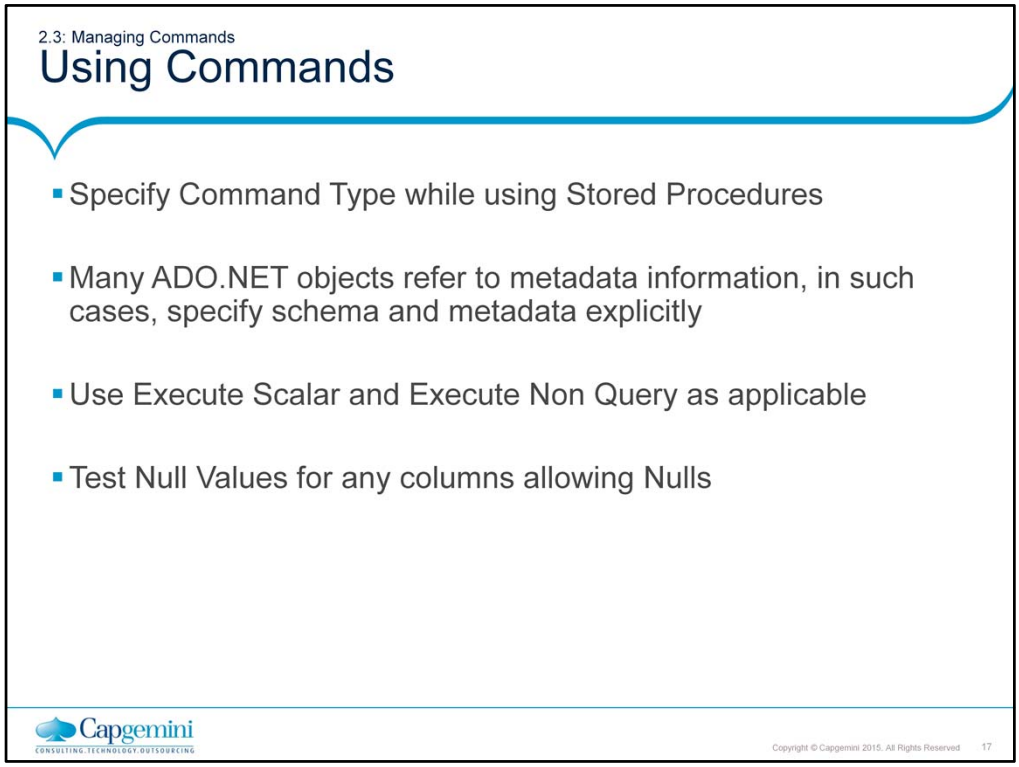
2.2: Commands to Manipulate Data]

Demo

- Using SQL Command
  - Manipulating Data using Sql Command Object
  - Manipulating Data using Sql Command Object along with Sql Parameter

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 16



Managing Commands:
ADO. NET provides different methods for command execution and various options for optimizing the execution of a command.
Some noteworthy points when working with Commands are as follows:

If a Stored Procedure is being called, then specify a CommandType property SQLCommand to StoredProcedure. This eliminates the need to parse the command before execution, since it is explicitly identified as stored procedure.

Many ADO.NET objects refer to metadata information, for such objects specify the metadata explicitly.

For example: The DataAdapter.Fill creates a table and columns in DataSet if none exist. CommandBuilder generates DataAdapter command properties for single-table SELECT commands. There is a performance hit each time this feature is used.

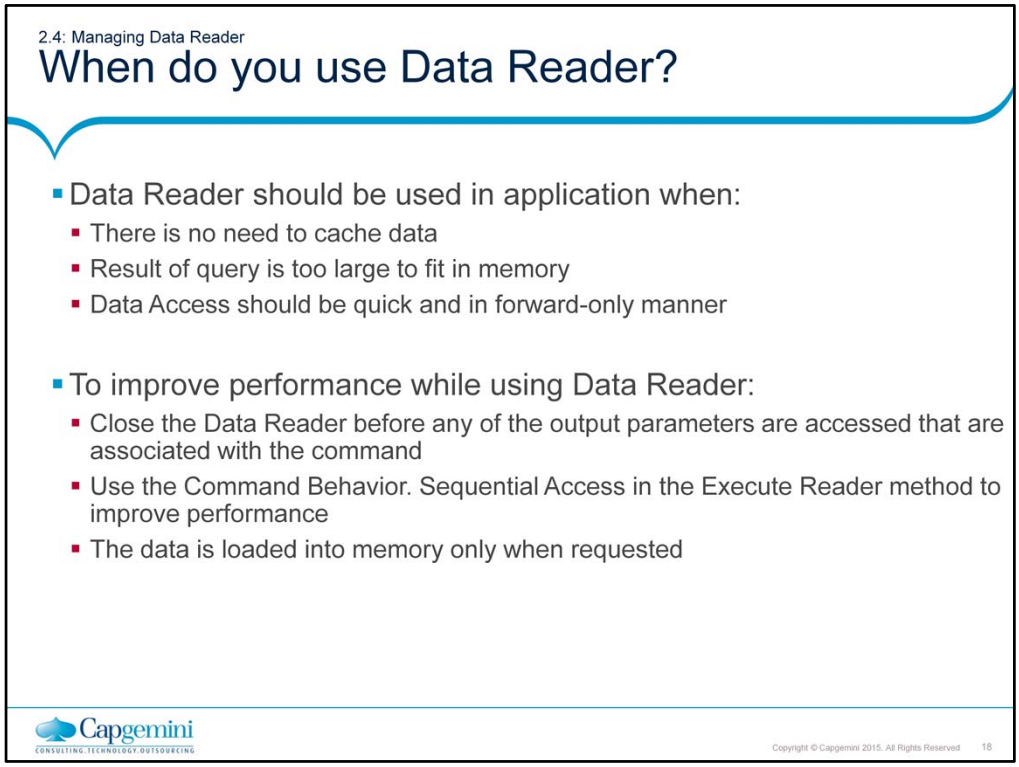
Always use ExecuteScalar method when the query returns a single value.

Whenever using DMLs, use ExecuteNonQuery. This avoids unnecessary processing to create an empty DataReader.

Test the null values for any columns allowing nulls. You cannot check a parameter value equal to null. Instead you need to write a WHERE clause which will test both cases – when the column is null and parameter is null.

For example:

```
SELECT * FROM Customers
WHERE (CompanyName=@companyname) OR
(CompanyName IS NULL AND @companyname IS NULL)
```

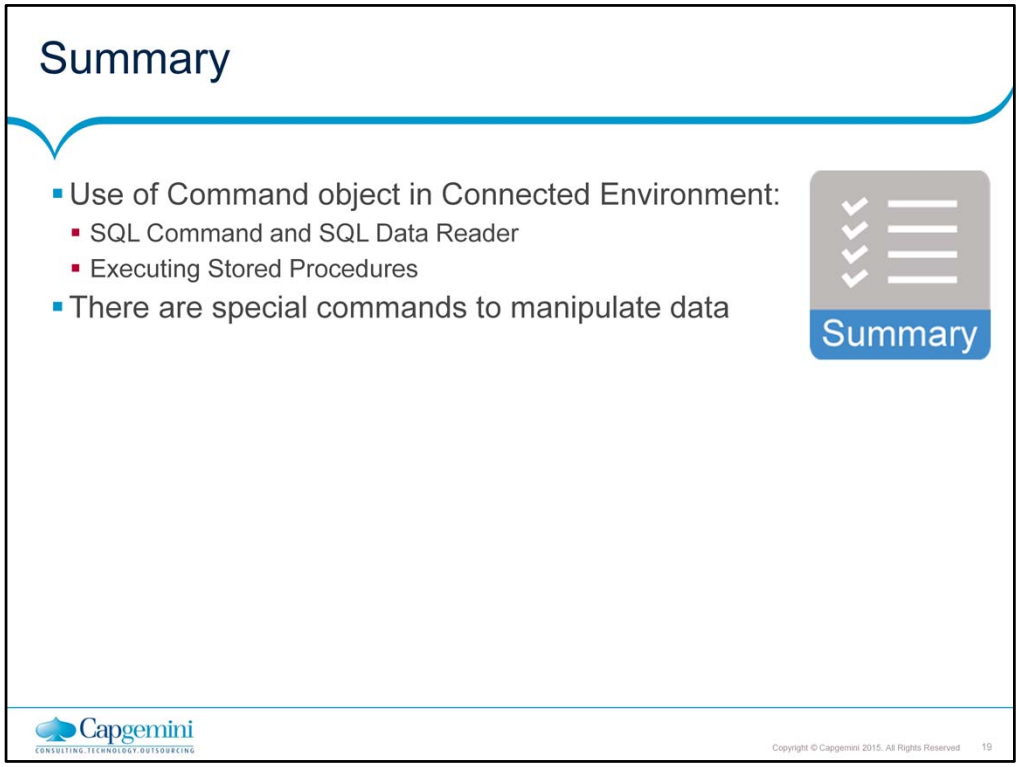


Managing DataReader:

When do you use DataReader?

As mentioned on the above slide, you can use Data Reader in your application. Also some additional set of recommendations for improving performance while using Data Reader, are as follows:

Close the DataReader before any of the output parameters are accessed that are associated to the Command. Also once you have finished reading the data close the DataReader.
CommandBehavior.SequentialAccess : It provides a way for the DataReader to handle rows that contain columns with large binary or string values. Rather than loading the entire row, SequentialAccess enables the DataReader to load data as a stream. You can then use the GetBytes or GetChars method to specify a byte location to start the read operation, and a limited buffer size for the data being returned.When you specify SequentialAccess, you are required to read from the columns in the order they are returned, although you are not required to read each column. Once you have read past a location in the returned stream of data, data at or before that location can no longer be read from the DataReader.



Add the notes here.

Review Question

- Question 1: The _____ method builds a SQL Data Reader object.
- Question 2: The property that allows to specify the type of command to execute
  - Option A: Cmd Type
  - Option B: Command Type
  - Option C: Type
- Question 3: The Begin Transaction () method is associated to _____.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 20

Add the notes here.

ADO.NET 4.5

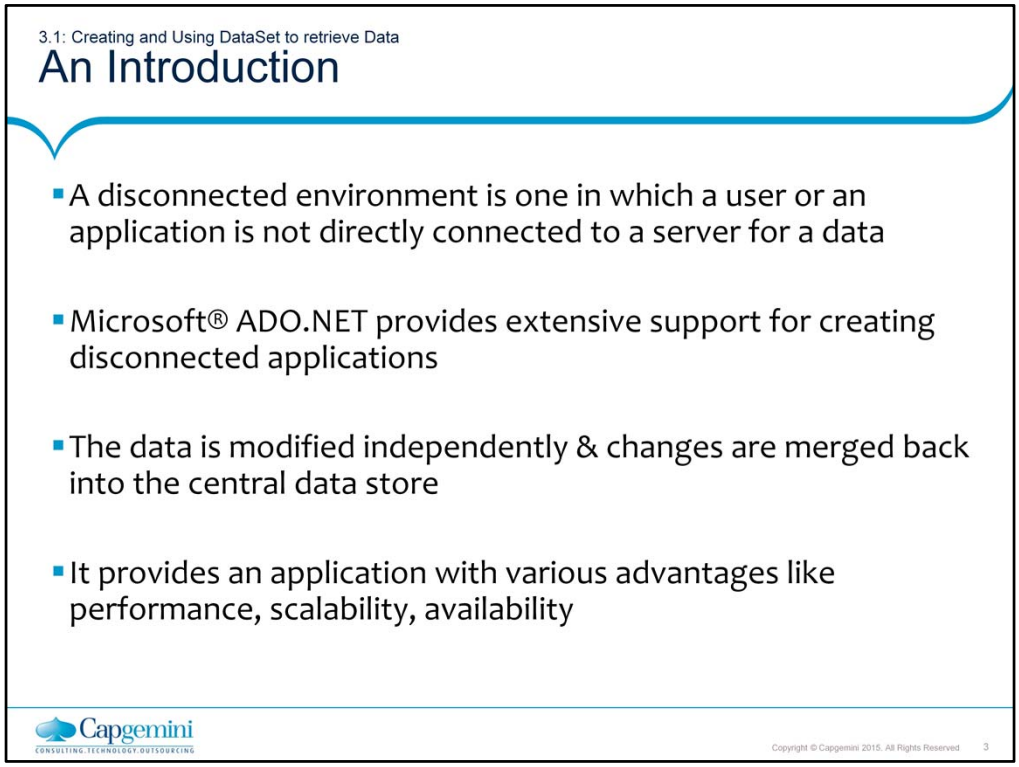
Lesson 3: Working with disconnected architecture

Lesson Objectives

- In this lesson, you will learn:
  - Creation and use of Data Set to retrieve data
  - Manipulation of database using Data Set
  - Implementing Transactions in ADO.NET

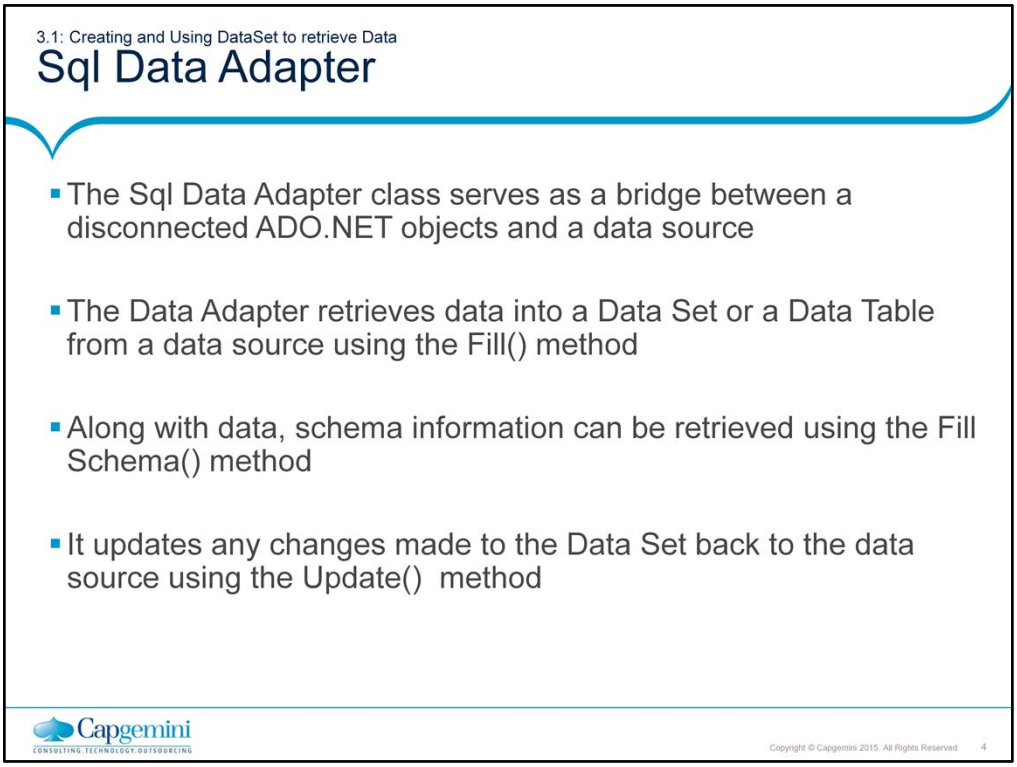
Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 2



Introduction :-

This lesson describes how to retrieve data by using Microsoft ADO.NET and how to disconnect from the data source, work with that data, and then reconnect to the ADO.NET data source to save any data updates. The lesson also describes how to create a DataSet object programmatically, and explains how to populate and save data in a DataSet by using a DataAdapter. The lesson also describes how to create a customized view of the data in a DataSet by using a DataView object.



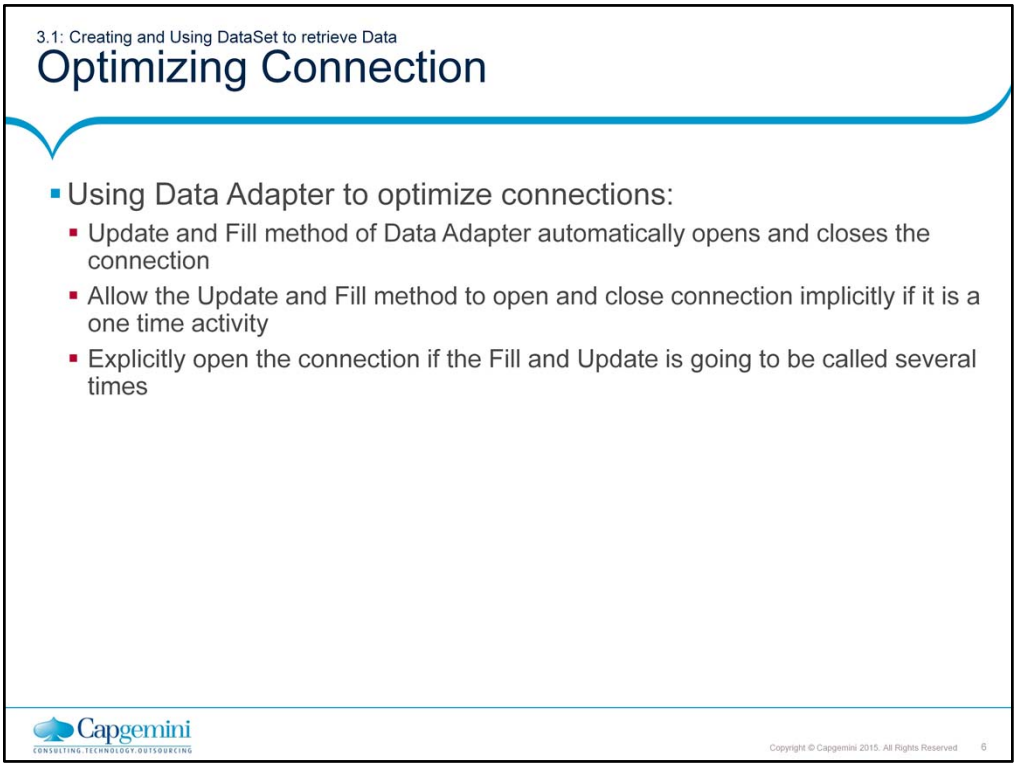
SQLDataAdapter:
The SqlDataAdapter uses the Fill method , which changes the data in the DataSet to match the data in the data source, and the Update method, which changes the data in the data source to match the data in the DataSet, using the appropriate Transact-SQL statements against the data source.
In simple scenarios, the updating logic that the DataAdapter uses to reconcile changes made to the DataSet can be generated automatically from the query used to retrieve the data by using a CommandBuilder object. For more complex scenarios, custom update logic can be written to control the logic the DataAdapter uses while updating data. The update is performed on a row-by-row basis. For every inserted, modified, and deleted row, the Update method determines the type of change that has been performed on it whether it was an Insert, Update, or Delete operation. Depending on the type of change, the Insert, Update, or Delete command template executes to propagate the modified row to the data source. When the SqlDataAdapter fills a DataSet , it creates the necessary tables and columns for the returned data if they do not already exist. However, primary key information is not included in the implicitly created schema unless the MissingSchemaAction property is set to AddWithKey. You may also have the SqlDataAdapter create the schema of the DataSet , including primary key information, before filling it with data using FillSchema method. SqlDataAdapter is used in conjunction with SqlConnection and SqlCommand to increase performance when connecting to a SQL Server database. The SqlDataAdapter also includes the SelectCommand, InsertCommand, DeleteCommand, UpdateCommand, and TableMapping properties to facilitate the loading and updating of data.

SQLDataAdapter [Contd.]:

The **SelectCommand** gets or sets a **Transact-SQL** statement or stored procedure used to select records in the data source**.** When **SelectCommand** is assigned to an existing **SqlCommand**, the **SqlCommand** is not duplicated. The **SelectCommand** maintains a reference to the previously created **SqlCommand** object. If the **SelectCommand** does not return any rows, no tables are added to the DataSet, and no exception is raised.
Similarly the **InsertCommand**, **DeleteCommand** and **UpdateCommand** gets or sets a Transact-SQL statement or stored procedure to insert, delete and update records into the data source respectively. In the same way as the **SelectCommand** even the **InsertCommand**, **DeleteCommand** & **UpdateCommand** if assigned to an existing **SqlCommand**, the **SqlCommand** is not duplicated. It maintains a reference to the previously created **SqlCommand** object
When an instance of **SqlDataAdapter** is created, the read/write properties are set to some initial values. The **InsertCommand**, **DeleteCommand**, and **UpdateCommand** are generic templates that are automatically filled with individual values from every modified row through the parameters mechanism.
For every column that you propagate to the data source on Update, a parameter should be added to the **InsertCommand**, **UpdateCommand**, or **DeleteCommand**. The **SourceColumn** property of the **DbParameter** object should be set to the name of the column. This setting indicates that the value of the parameter is not set manually, but is taken from the particular column in the currently processed row.

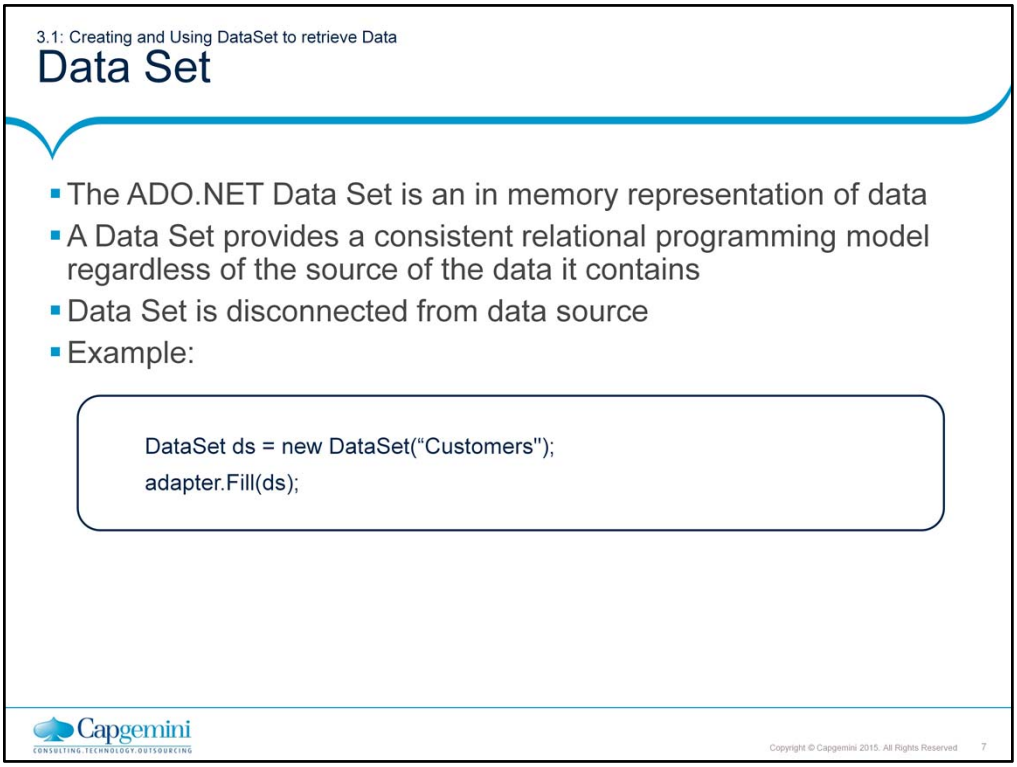
The **ContinueUpdateOnError** property of dataadapter controls whether an Update() continues with remaining rows or stops processing if an error is encountered during the updating. If ContinueUpdateOnError is true, and an error is encountered during the update, an exception isn't raised, the RowError property of the DataRow causing the error is set to the error message that would have been raised, and the update continues processing the remaining rows. A well-designed application uses the RowError information to present the user with a list of the failed and possibly the current values in the data source for those rows. It also provides a mechanism to correct and resubmit the failed attempts, if required.

If ContinueUpdateOnError is false, the DataAdapter raises a DBConcurrencyException when a row update attempt fails. Generally, ContinueUpdateOnError is set to false when the changes made to the DataSet are part of a transaction and must be either completely applied to the data source or not applied at all. The exception handler rolls back the transaction.



Managing Connections:

The DataAdapter's Fill and Update method automatically opens the connection, executes, and closes the connection. However, it is recommended that if it is a one time activity, then allow the Update and Fill method to implicitly open and close connection. Else you open the connection explicitly, that is if the methods are to be called several times. Close the connection once the required Fill and Update methods have executed.
Also while performing transaction, open the connection and once the transaction is completed, commit your work and then close the connection.



DataSet:

The DataSet is a memory-resident representation of data that provides consistent relational programming model regardless of the source of the data it contains. A DataSet represents a complete set of data including the tables that contain columns, rows and constrain the data, as well as the relationships between the tables. It is disconnected from the data source.

The principal class used by the ADO.NET disconnected model is System.Data.DataSet. A DataSet object provides an in-memory cache of data. The data in a DataSet is held in a format that is independent of any underlying data source.

There are several methods of working with a DataSet, which can be applied independently or in combination. You can:

Programmatically create DataTables, DataRelations, and Constraints within the DataSet and populate the tables with data.
Populate the DataSet with tables of data from an existing relational database management system using a DataAdapter.
Load and persist the DataSet contents using XML.

DataSet [Contd.]:

In a typical multi-tier implementation, the steps for creating and refreshing a **DataSet**, and in turn, updating the original data are to:

- Build and fill each **DataTable** in a **DataSet** with data from a data source using a **DataAdapter**.
- Change the data in individual **DataTable** objects by adding, updating, or deleting **DataRow** objects.
- Invoke the **GetChanges** method to create a second **DataSet** that features only the changes to the data.

Call the **Update** method of the **DataAdapter**, passing the second **DataSet** as an argument.

- Invoke the **Merge** method to merge the changes from the second DataSet into the first.
- Invoke the **AcceptChanges** on the DataSet. Alternatively, invoke **RejectChanges** to cancel the changes

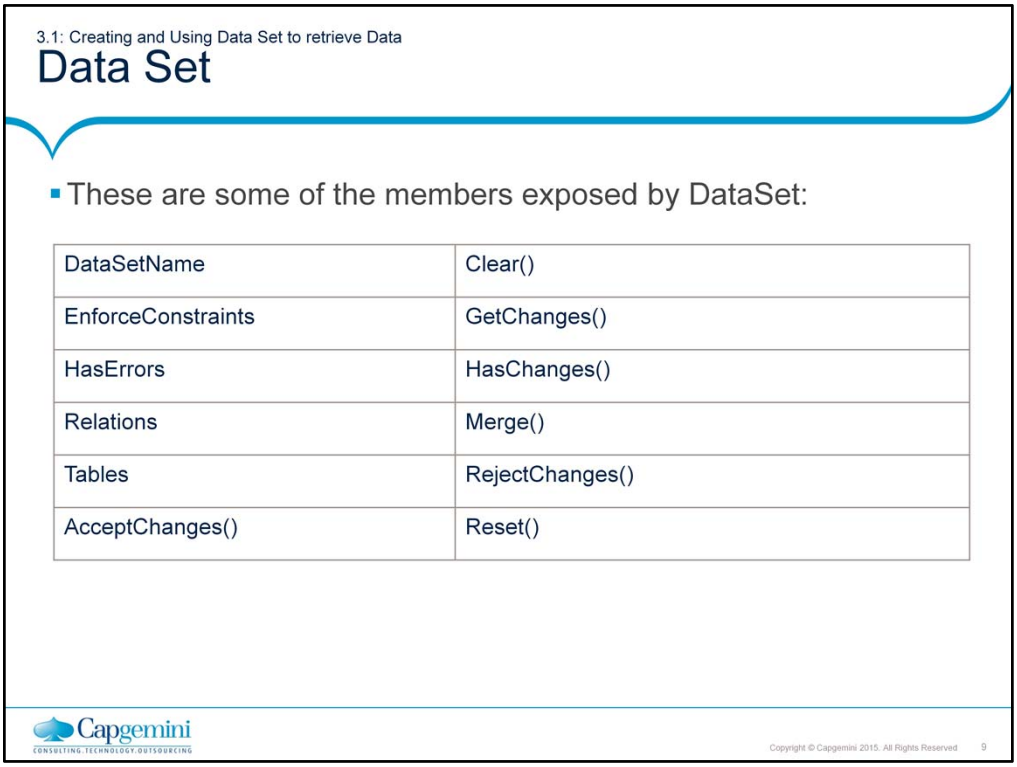
The design of the ADO.NET DataSet makes this scenario easy to implement. Because the DataSet is stateless, it can be safely passed between the server and the client without tying up server resources such as database connections. Although the DataSet is transmitted as XML, Web Services and ADO.NET automatically transform the XML representation of the data to and from a DataSet, creating a rich, yet simplified, programming model. Additionally, because the DataSet is transmitted as an XML stream, non-ADO.NET clients can consume the same Web Service as that consumed by ADO.NET clients. Similarly, ADO.NET clients can interact easily with non-ADO.NET Web Services by sending any client DataSet to a Web Service as XML data and by consuming any XML returned as a DataSet from the Web Service.

The DataSet consists of a collection of DataTable objects that you can relate to each other with DataRelation objects. You can also enforce data integrity in the DataSet by using the **UniqueConstraint** and **ForeignKeyConstraint** objects.

Whereas DataTable objects contain the data, the **DataRelationCollection** allows you to navigate though the table hierarchy. The tables are contained in a **DataTableCollection** accessed through the Tables property. When accessing **DataTable** objects, note that they are conditionally case sensitive.

A DataSet can read and write data and schema as XML documents. The data and schema can then be transported across HTTP and used by any application, on any platform that is XML-enabled. You can save the schema as an XML schema with the **WriteXmlSchema** method, and both schema and data can be saved using the WriteXml method. To read an XML document that includes both schema and data, use the **ReadXml** method.

(Note: We will be elaborating on Data Table and Data Relation later in this lesson)



DataSet Members:
These are some members of DataSet:

DataSetName - Gets or sets the name of the current DataSet
EnforceConstraints – Gets or sets a value indicating whether constraint rules are followed when attempting any update operation
HasErrors – Indicates whether there are errors in any of the rows in any of the tables of this DataSet
Relations – Retrieves the collection of relations that link tables and allow navigation from parent tables to child tables
Tables - Retrieves the collection of tables contained in the DataSet.
AcceptChanges() – Commits all the changes made to this DataSet since it was loaded or the last time AcceptChanges was called
Clear()- clears the DataSet of any data by removing all rows in all tables.
GetChanges()- Retrieves a copy of the DataSet containing all changes made to it since it was last loaded, or since AcceptChanges was called.
HasChanges()- Retrieves a value indicating whether the DataSet has changes, including new, deleted, or modified rows
RejectChanges()- Rolls back all the changes made to the DataSet since it was created, or since the last time DataSet.AcceptChanges was called
Merge() – Merges this DataSet with a specified DataSet.
Reset()- Resets the DataSet to its original state. Subclasses should override Reset to restore a DataSet to its original state.

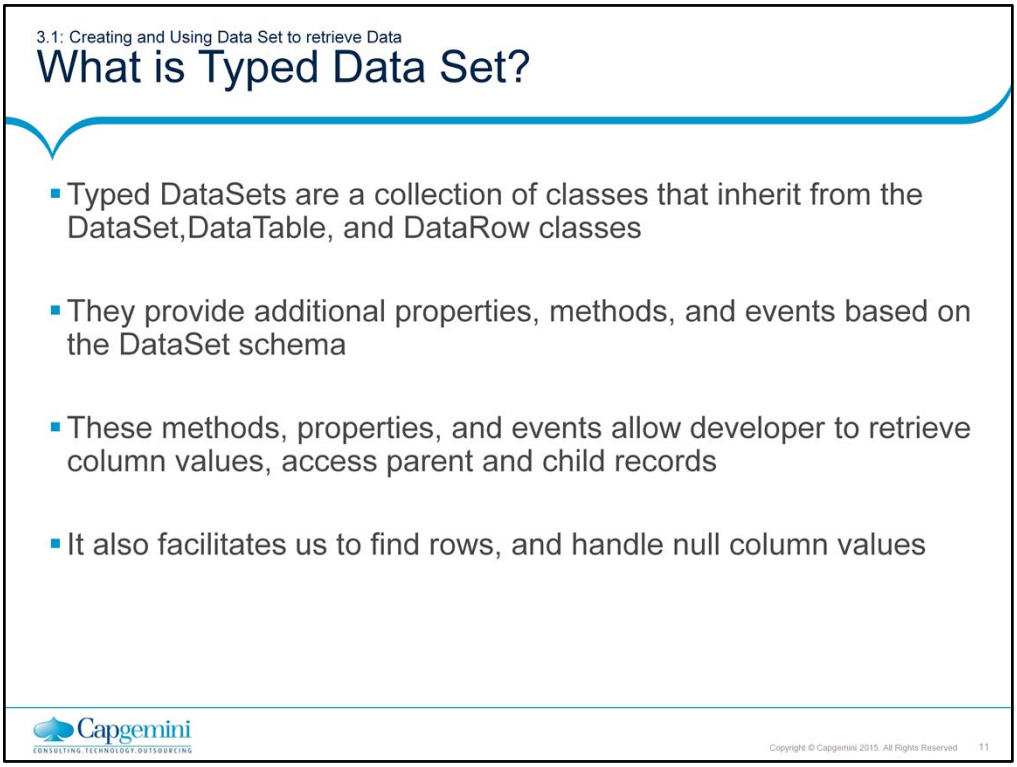
3.1: Creating and Using DataSet to retrieve Data

Demo

- Using DataSet and DataAdapter

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 10



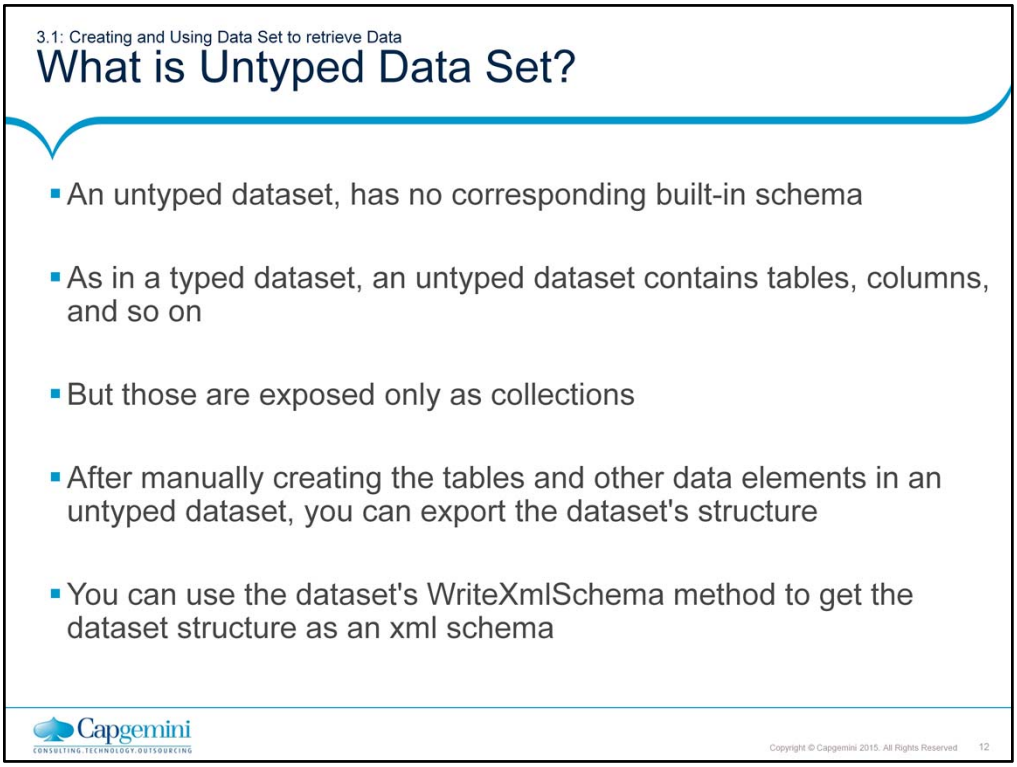
What is Typed DataSet?

A typed dataset is a dataset that is first derived from the base DataSet class and then uses information from the Dataset Designer, which is stored in an .xsd file, to generate a new strongly-typed dataset class. Information from the schema (tables, columns, and so on) is generated and compiled into this new dataset class as a set of first-class objects and properties. Because a typed dataset inherits from the base DataSet class, the typed class assumes all of the functionality of the DataSet class and can be used with methods that take an instance of a DataSet class as a parameter.

There are some drawbacks to using strongly typed DataSet objects:

Using typed classes adds some overhead to code execution. If the strongly typed functionality isn't required, application performance can be improved slightly using an untyped DataSet.

The strongly typed DataSet will need to be regenerated when the data structure changes. Applications using the typed DataSet will need to be rebuilt using a reference to the new typed DataSet. This can be especially significant in a multitier application or distributed where any clients that use the typed DataSets will have to be rebuilt using a reference to the updated version, even those that would not otherwise have been affected by the change if an untyped DataSet was used.

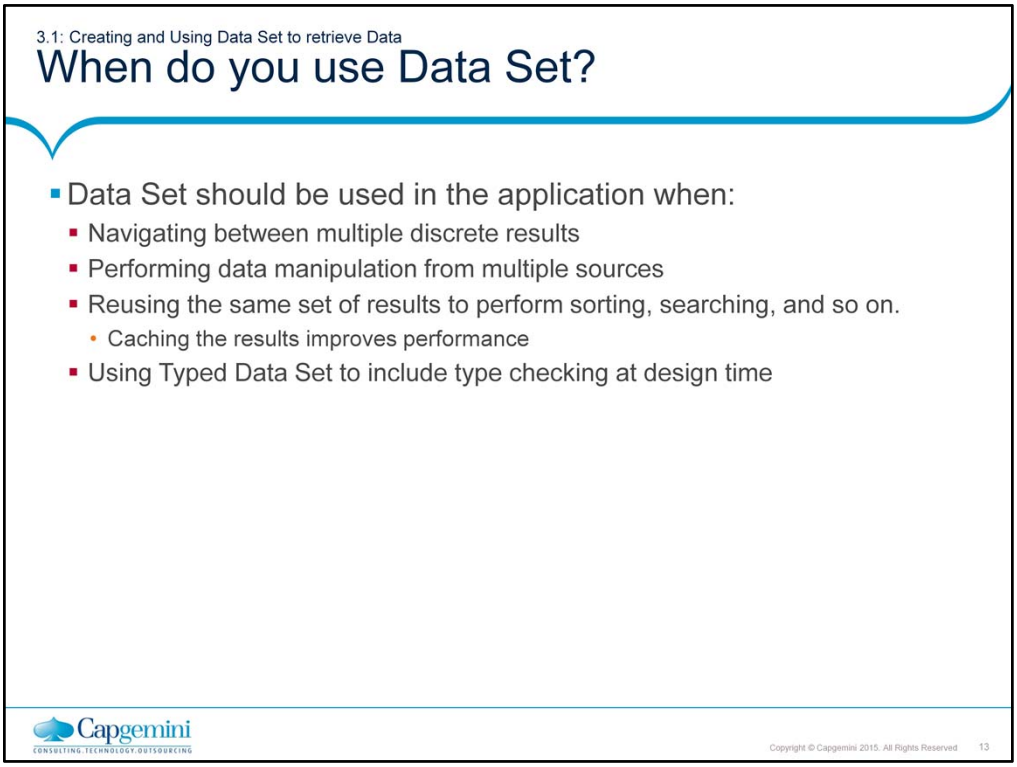


Difference between Typed DataSet and Untyped DataSet

Typed DataSet
1. Typed DataSets are DataSets with XML Schema Definition (xsd).
2. They perform error checking regarding their schema at design time using the .xsd definitions.
3. Typed DataSets have intelliSense support..
4. Performance is slower in case of strongly typed dataset.
5. In complex environment, strongly typed dataset's are difficult to administer.

Untyped DataSet

1. Untyped DataSets are DataSets without XML Schema Definition (xsd).
2. Erros cannot be trapped at the design time as they are filled at run time when the code executes.
3. Do not support intelliSense.
4. Performance is faster in case of Untyped dataset.
5 .Untyped datasets are easy to administer.



Managing DataSets and Other Objects:

When do you use DataSet?

DataSet can be used in the application whenever discrete table of results is being navigated and also if the application manipulates data from different sources like a relational database or XML file.
Additionally, caching the data using the DataSet improves performance for tasks such as Sorting, Filtering.
To include type checking at design time use Typed DataSet. Apart from this in .Net environment statement completion is also supported when you work with Typed DataSet.

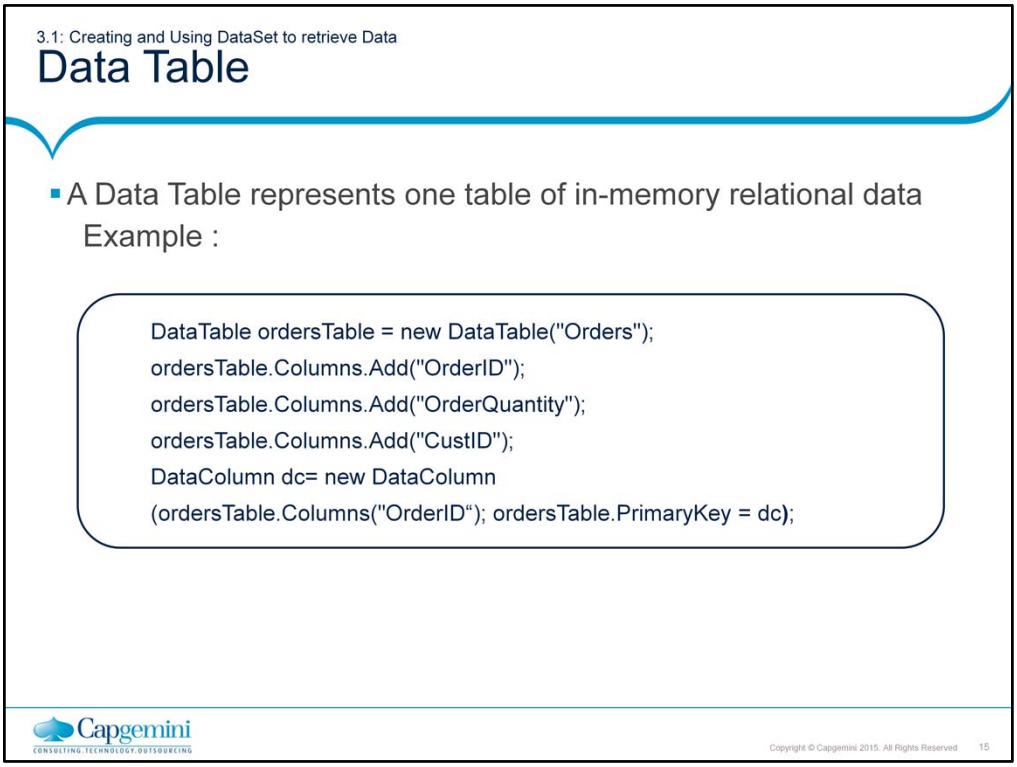
Some more noteworthy points while working with DataSet:

To Refresh Data in DataSet: To get updated values from the server use the DataAdapter.Fill method. This method matches new rows based on primary keys and applies corresponding changes from the server, if a Primary Key is defined for the DataTable. If the Primary Key is not defined, then the Fill method adds new rows for the refreshed values, which might lead to addition of duplicate rows. If you want to retain the changes made to the rows in the table and also refresh the data from the server then you must first populate it with DataAdapter.Fill method, and fill a new DataTable. Then merge that DataTable into the DataSet by setting preserveChanges value to true.
To Search Data in a DataSet: While querying DataSet based on a criteria, the performance can be improved by taking advantage of index-based lookups. An index is created, whenever a Primary Key is assigned to the DataTable, or a DataView is added for a DataTable.

Managing DataSets and Other Objects:

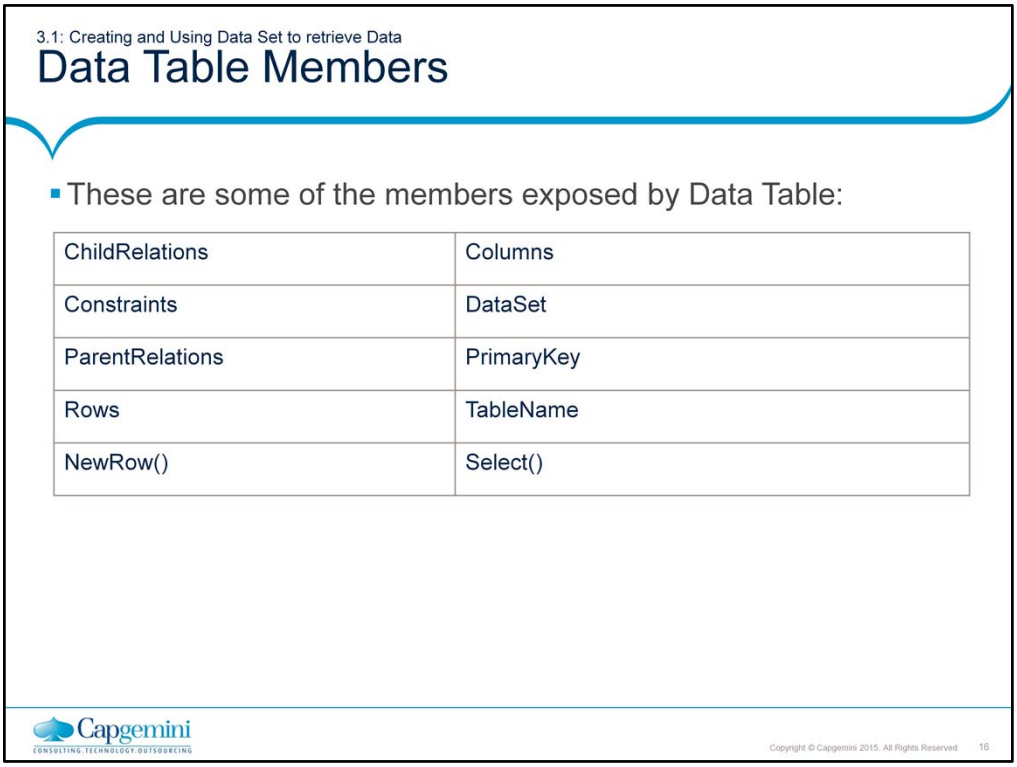
When do you use DataSet (contd.)?

- If the query is fired on the Primary Key columns of the DataTable, then use **DataTable.Rows.Find**, rather than using **DataTable.Select**.
- Specify a sort order for the DataView, so that an index is available for queries involving Non-primary Key columns.
- However, remember to use the **index** feature only when you are performing multiple queries. Creating the index is costly and overall benefit of using the index is reduced.



DataTable:

A DataTable, which represents one table of in-memory relational data, can be created and used independently, or can be used by other .NET Framework objects, most commonly as a member of a DataSet.
You can create a DataTable object by using the DataTable constructor, or by passing constructor arguments to the Add method of the Tables property of the DataSet, which is a DataTableCollection.
When accessing DataTable objects, note that they are conditionally case sensitive. For example, if one DataTable is named "mydatatable" and another is named "Mydatatable", a string used to search for one of the tables is regarded as case sensitive. However, if "mydatatable" exists and "Mydatatable" does not, the search string is regarded as case insensitive. A DataSet can contain two DataTable objects that have the same TableName property value but different Namespace property values.
You can also create DataTable objects within a DataSet by using the Fill or FillSchema methods of the DataAdapter object. If you are creating a DataTable programmatically, you must first define its schema by adding DataColumn objects to the DataColumnCollection which can be accessed through the Columns property.
To add rows to a DataTable, you must first use the NewRow method to return a new DataRow object. The NewRow method returns a row with the schema of the DataTable, as it is defined by the table's DataColumnCollection. The maximum number of rows that a DataTable can store is 16,777,216. The DataTable also contains a collection of Constraint objects that can be used to ensure the integrity of the data.



DataTable Members & Events:

DataTable dTable = new DataTable("Customers“)

The following example creates an instance of a DataTable by adding it to the Tables collection of a DataSet.

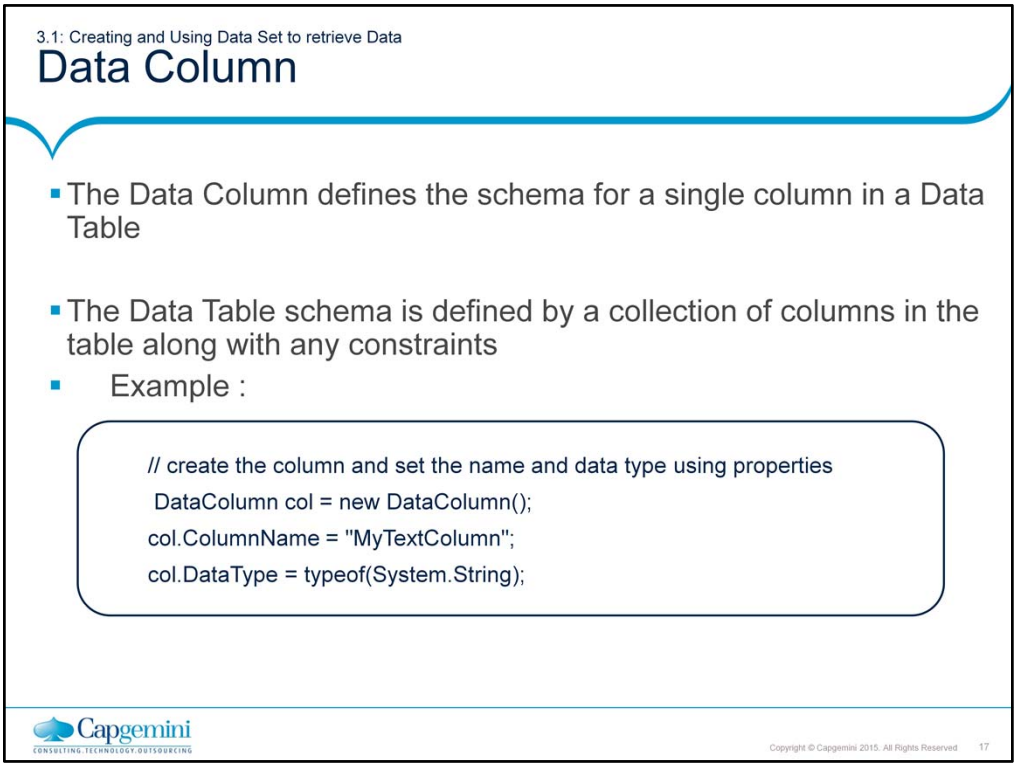
```
DataSet custDS = new DataSet();
DataTable custTable = custDS.Tables.Add("CustTable")
```

Some of the important members of DataTable object.

ChildRelations - Gets the collection of child relations for this DataTable.
Columns- Gets the collection of columns that belong to this table.
Constraints - Gets the collection of constraints maintained by this table.
DataSet - Gets the DataSet that this table belongs to.
ParentRelations - Gets the collection of parent relations for this DataTable.
PrimaryKey - Gets or sets an array of columns that function as primary keys for the data table.
Rows - Gets the collection of rows that belong to this table.
TableName - Gets or sets the name of the DataTable.
NewRow() - This method creates a new DataRow with the same schema as the table.
Select()- This method Gets an array of DataRow objects.

The DataTable also exposes some events as:

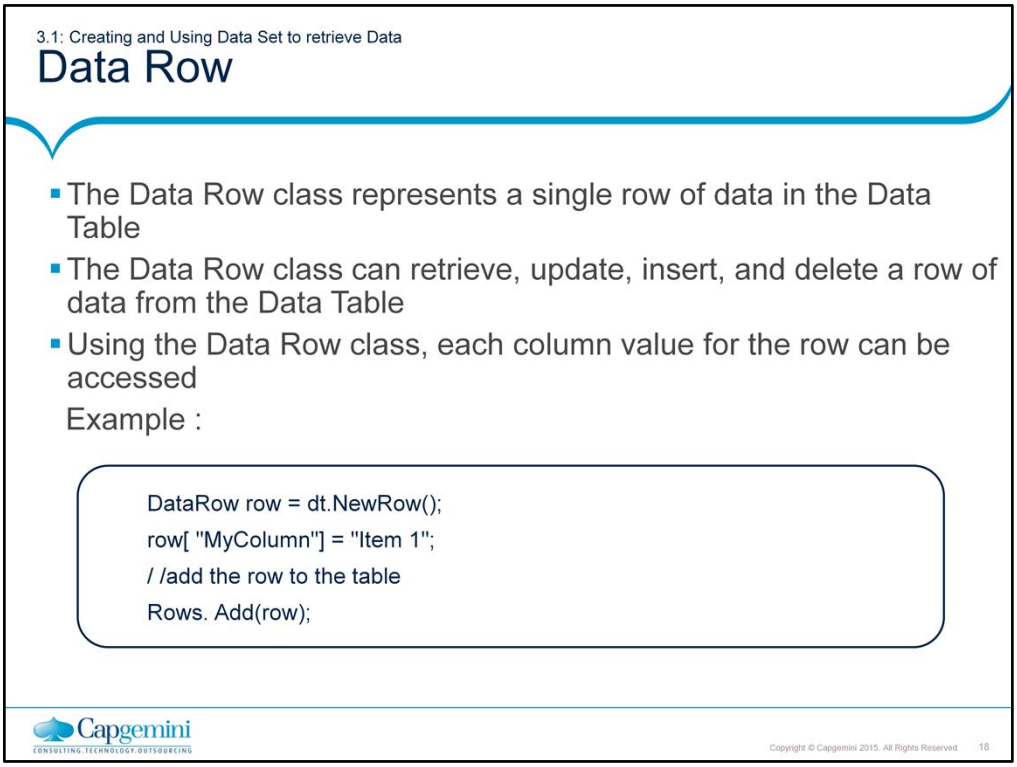
ColumnChanged - Occurs when after a value has been changed for the specified DataColumn in a DataRow.
ColumnChanging - Occurs when a value is being changed for the specified DataColumn in a DataRow.
RowChanged - Occurs after a DataRow has been changed successfully.
RowChanging - Occurs when a DataRow is changing.
RowDeleted - Occurs after a row in the table has been deleted.
RowDeleting - Occurs before a row in the table is about to be deleted



DataColumn

The DataColumn defines the schema for a single column in a DataTable. The DataTable schema is defined by a collection of columns in the table along with any constraints. The DataColumn defines:

The type of data that can be stored in the column
The length of the column for text-based column types
Whether the data in the column can be modified
Whether the column values for each row must be unique
Whether the column in rows can contain null values
Whether the column values are automatically generated and the rules for generating those values
Whether the column value is calculated based on an expression



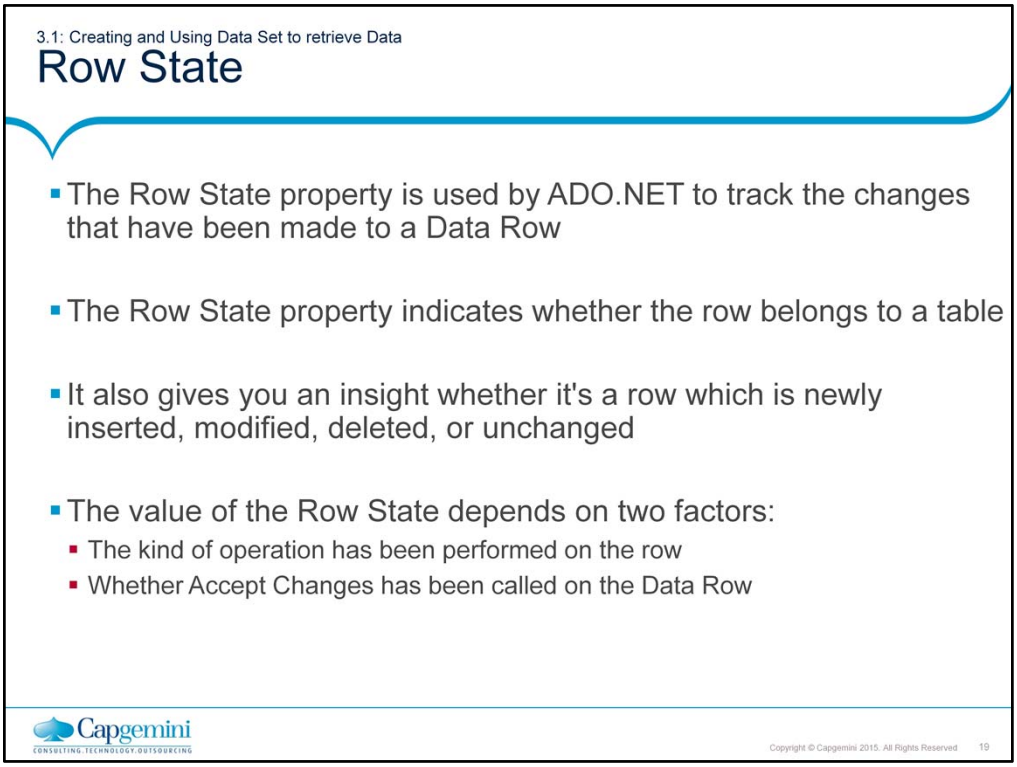
DataRow

The DataRow class represents a single row of data in the DataTable. The DataRow class can retrieve, update, insert, and delete a row of data from the DataTable. Using the DataRow class, each column value for the row can be accessed. The DataRow maintains the RowState property that is used by ADO.NET to track the changes that have been made to a DataRow. This property allows changed rows to be identified, and the appropriate update command to be used to update the data source with the changes.

Creating a DataRow

A DataRow is created by calling the NewRow() method of a DataTable, a method that takes no arguments. The DataTable supplies the schema, and the new DataRow is created with default or empty values for the fields:

```
// create a table with one column
DataTable dt = new DataTable();
dt.Columns.Add("MyColumn", typeof(System. String));

//create a row with the same schema as DataTable
dtDataRow row = dt.NewRow();
row[ "MyColumn"] = "Item 1";
// add the row to the table
dt. Rows.Add(row);
```



RowState

The RowState property is used by ADO.NET to track the changes that have been made to a DataRow, which allows changes made to the data while disconnected to be updated back to the data source. The RowState property indicates whether the row belongs to a table, and if it does, whether it's newly inserted, modified, deleted, or unchanged since it was loaded.
The value of the RowState property can't be set directly. ADO.NET sets the row state in response to actions that affect the DataRow. The AcceptChanges() and RejectChanges() methods, whether explicitly or implicitly called, both reset the RowState value for the row to Unchanged.
ADO.NET maintains up to three versions of each DataRow object. The DataAdapter reconciles changes made since the data was loaded from the data source, thereby making changes to the disconnected data permanent. Two versions of each row are maintained to allow the DataAdapter to determine how to perform the reconciliation. The Original version contains the values that were loaded into the row. The Current version contains the latest version of the data, including the changes made since the data was originally loaded. The Original version isn't available for newly created rows. ADO.NET also allows a row to be put into edit mode which temporarily suspends events for the row and allows the user to make multiple changes to the row without triggering validation rules. The BeginEdit( ) method of the DataRow puts the row into edit mode, while the EndEdit( ) and CancelEdit( ) methods take the row out of edit mode. AcceptChanges( ) also takes the row out of edit mode because it implicitly calls EndEdit( ), as does RejectChanges( ), which implicitly calls CancelEdit( ).

RowState

The **RowState** does detect the whether a row has been changed or no. But sometimes you may need to access the information in a deleted row in a **DataTable**. If you access the deleted row directly, you will get an exception saying that the row has been deleted.

To avoid the exception, but still access the information in the deleted row of the DataTable, you just need to specify that you want to get the information from the original version of the row by specifying **DataRowVersion.Original** as follows:

```
if (dataRow.RowState == DataRowState.Deleted)
    id = (string)dataRow["CustomerID", DataRowVersion.Original];
```

You can get only the deleted rows from a DataTable by iterating through all the rows and checking the DataRowState, or you could simply create a DataView that filters on DataViewRowState.Deleted:

```
// Get Only Deleted Rows in DataTable
DataView dv = new DataView(sourceDataTable,
                           null, null, DataViewRowState.Deleted);
```

You can also convert that DataView to a DataTable using the new **DataView.ToTable()** method:

```
DataTable dt = dv.ToTable();
```

The rows in the new DataTable, dt, will not be marked as **DataRowState.Deleted** but rather **DataRowState.Added**, because this is a new DataTable. You can easily iterate through this table now as none of the rows will be mark deleted and you will have full access to the information without specifying **DataRowVersion**.

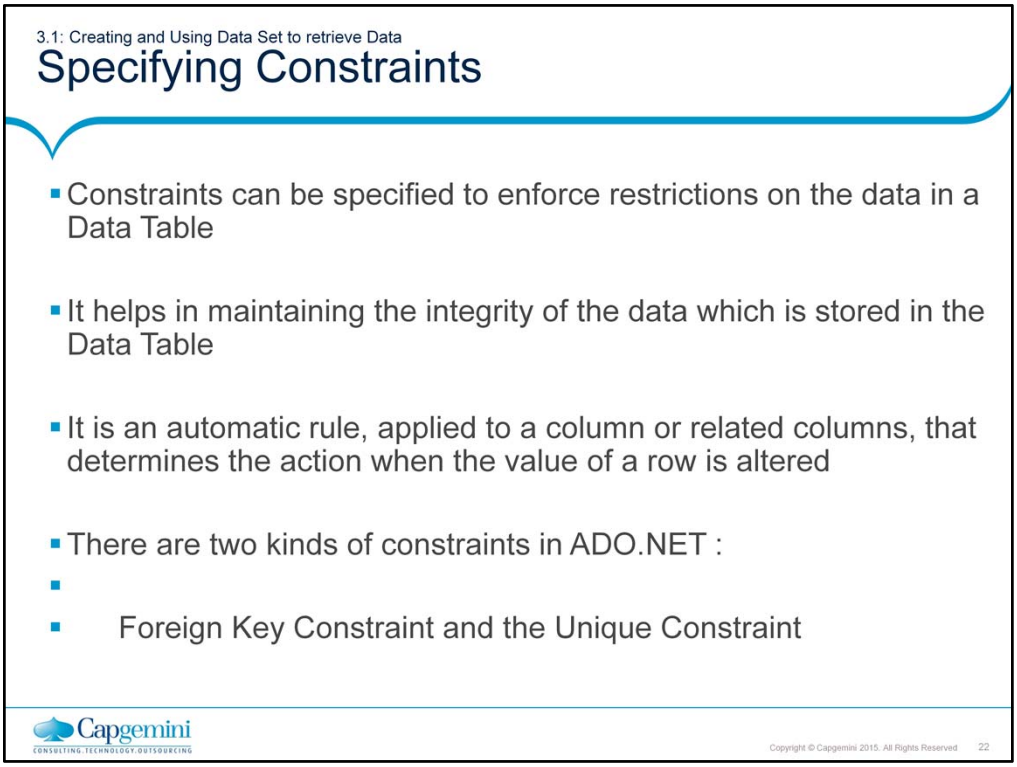
3.1: Creating and Using Data Set to retrieve Data

Demo

- Understanding Row State

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

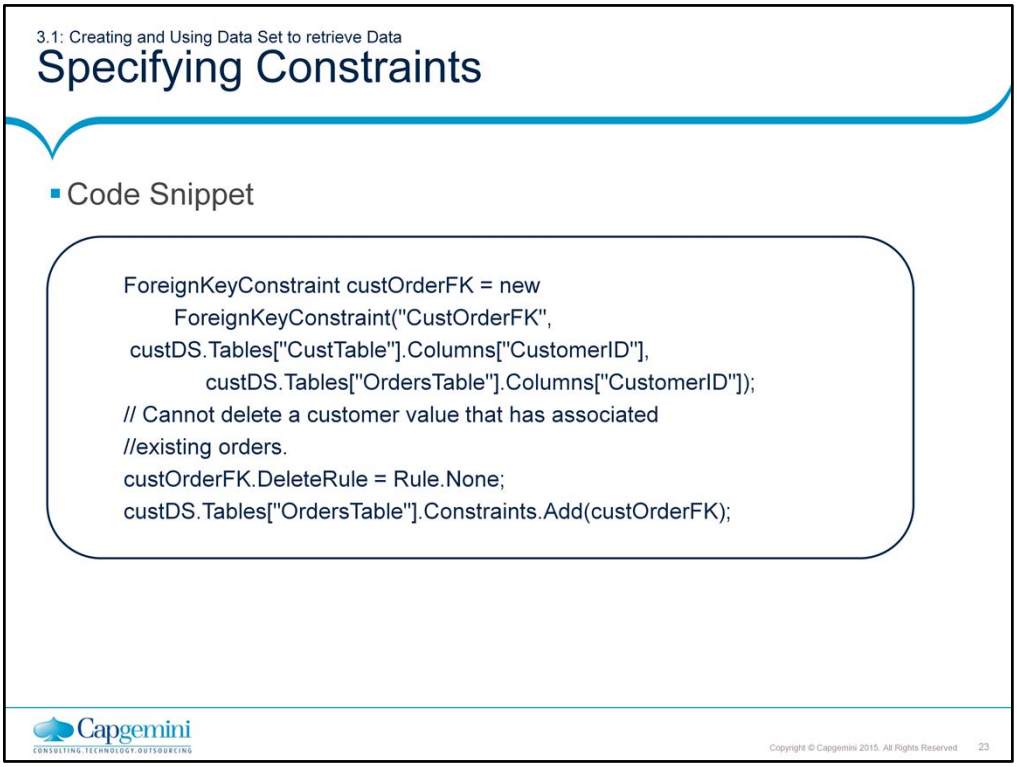
 21



Specifying Constraints

You can use constraints to enforce restrictions on the data in a DataTable, in order to maintain the integrity of the data. A constraint is an automatic rule, applied to a column or related columns, that determines the course of action when the value of a row is somehow altered. Constraints are enforced when the EnforceConstraints property of the DataSet is TRUE.
There are two kinds of constraints in ADO.NET: the ForeignKeyConstraint and the UniqueConstraint. By default, both constraints are created automatically when you create a relationship between two or more tables by adding a DataRelation to the DataSet. However, you can disable this behavior by specifying createConstraints = FALSE when creating the relation.
A ForeignKeyConstraint enforces rules about how updates and deletes to related tables are propagated
The DeleteRule and UpdateRule properties of the ForeignKeyConstraint define the action to be taken when the user attempts to delete or update a row in a related table. The settings for DeleteRule and UpdateRule properties for the ForeignKeyConstraint are as follows:-

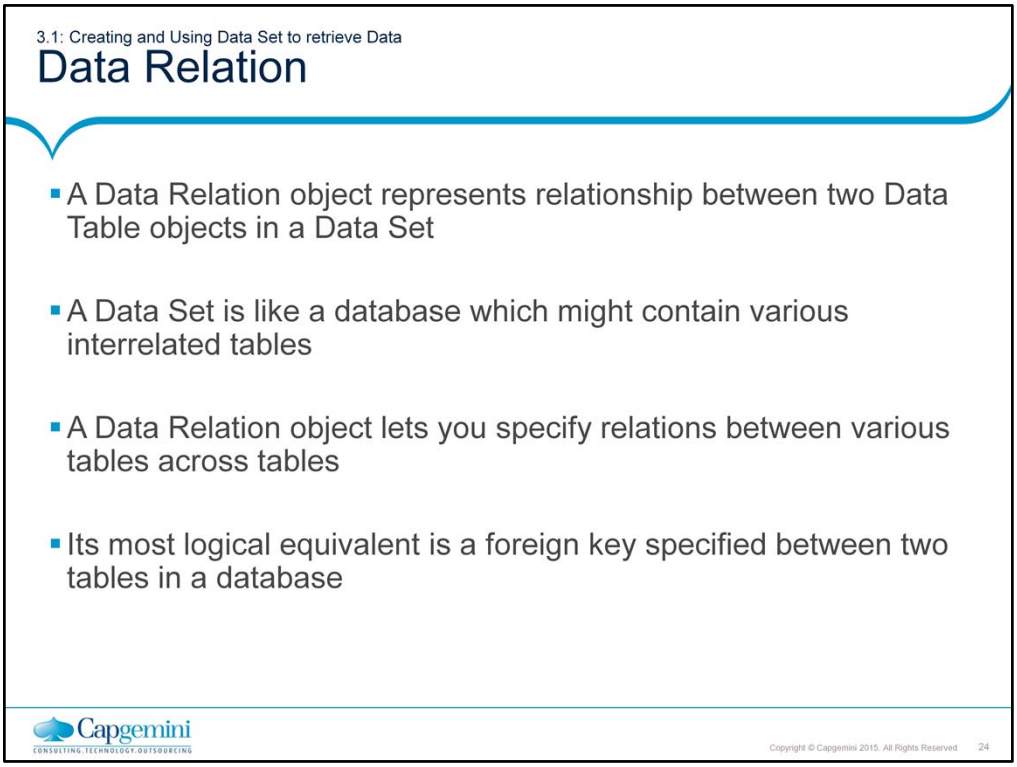
- **Cascade** - Deletes or updates related rows.
- **SetNull** - Set values in related rows to DBNull.
- **SetDefault** - Set values in related rows to the default value.
- **None** -Take no action on related rows. This is the default behavior.



Specifying Constraints [Contd]

The UniqueConstraint object, which can be assigned either to a single column or to an array of columns in a DataTable, ensures that all data in the specified column or columns is unique per row. You can create a unique constraint for a column or array of columns by using the constructor for UniqueConstraint. Pass the resulting UniqueConstraint object to the Add method of the table's Constraints property, which is a ConstraintCollection. When creating a UniqueConstraint for a column or columns, you can optionally specify whether the column or columns are a primary key.

You can also create a unique constraint for a column by setting the Unique property of the column to TRUE. Alternatively, setting the Unique property of a single column to false removes any unique constraint that may exist. Defining a column or columns as the primary key for a table will automatically create a unique constraint for the specified column or columns. If you remove a column from the PrimaryKey property of a DataTable, the UniqueConstraint is removed.

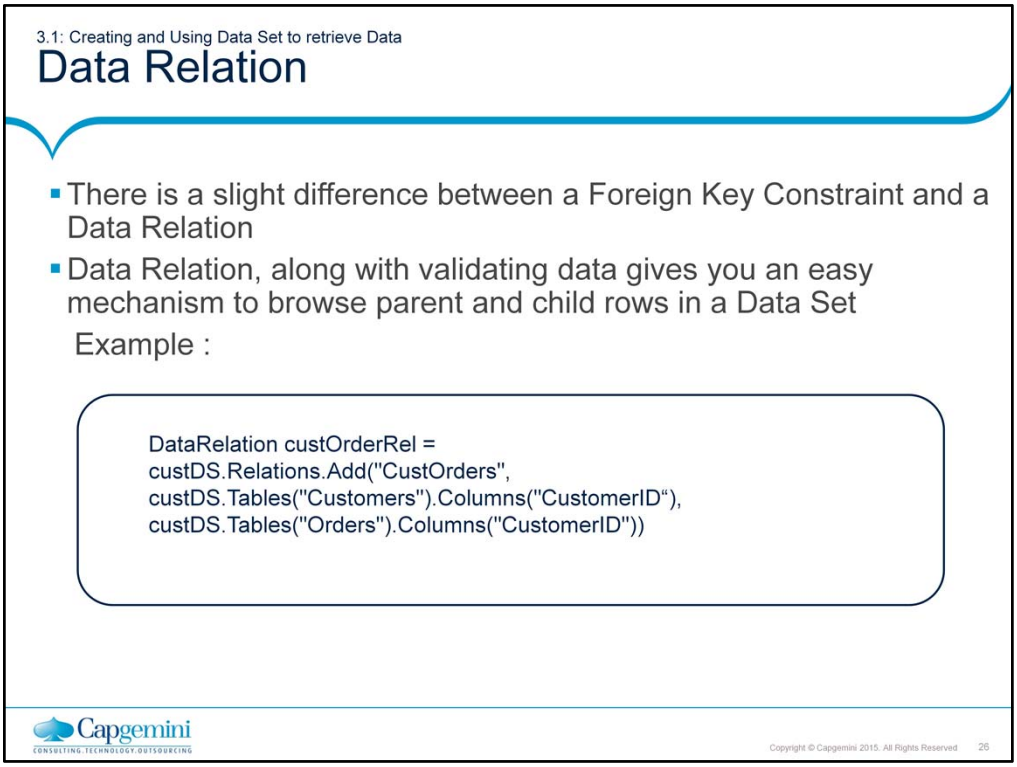


DataRelation

One of the biggest differences between traditional ADO and ADO.NET is that the rowsets stored within ADO.NET can be truly relational. For example, a DataSet can store one DataTable containing customers and another DataTable containing the customers' orders. These DataTable objects can then be related to one another within ADO.NET, thus recreating the relationship that exists within the relational database. In ADO.NET once you retrieve two rowsets of data (in other words, parents and children) and relate them to each other, you can then retrieve all children rows for a given parent, display any one DataTable in a grid at a time, or modify several tiers of DataTable objects, and send the changes to the database all in one batch update. DataRelation objects, which are integral to ADO.NET applications, enable these features to function.

In a DataSet that contains multiple DataTable objects, you can use DataRelation objects to relate one table to another, to navigate through the tables, and to return related values. For example, in a Customer/Orders relationship, the Customers table is the parent and the Orders table is the child of the relationship. This is similar to a primary key/foreign key relationship. Relationships are created between matching columns in the parent and child tables. That is, the DataType value for both columns must be identical.
The arguments required to create a DataRelation are the columns that serve as the parent and child columns in the relationship, and a name for the DataRelation being created. Once a DataRelation has been created, it can be used for navigating between two tables or when retrieving values.

When a DataRelation is created, it first verifies that the relationship can be established. After it is added to the DataRelationCollection, the relationship is maintained by disallowing any changes that would invalidate it. Between the period when a DataRelation is created and added to the DataRelationCollection, it is possible for additional changes to be made to the parent or the child rows. An exception is generated if this causes a relationship to become invalid.
DataRelation objects are contained in a DataRelationCollection, which you can access through the Relations property of the DataSet, and the ChildRelations and ParentRelations properties of the DataTable.



There are two reasons why you might define DataRelation objects:

To provide better error checking (for example, spotting an orphaned child before you reconnect to update the data source). This functionality is providedthrough the ForeignKeyConstraint, which the DataRelation can create implicitly.
To provide better navigation

Once a relation is established, you can use it navigate from a parent row to the associated child rows or from a child row to the parent. Use the GetChildRows() or GetParentRow() method of the DataRow object:

Example

```
foreach(DataRow parent in ds.Tables["Categories"].Rows)
 {
  //Process the category row
  foreach(DataRow child in parent.GetChildRows("Cat_Prod"))
 {
  //Process the products in this category
   }
   }
```

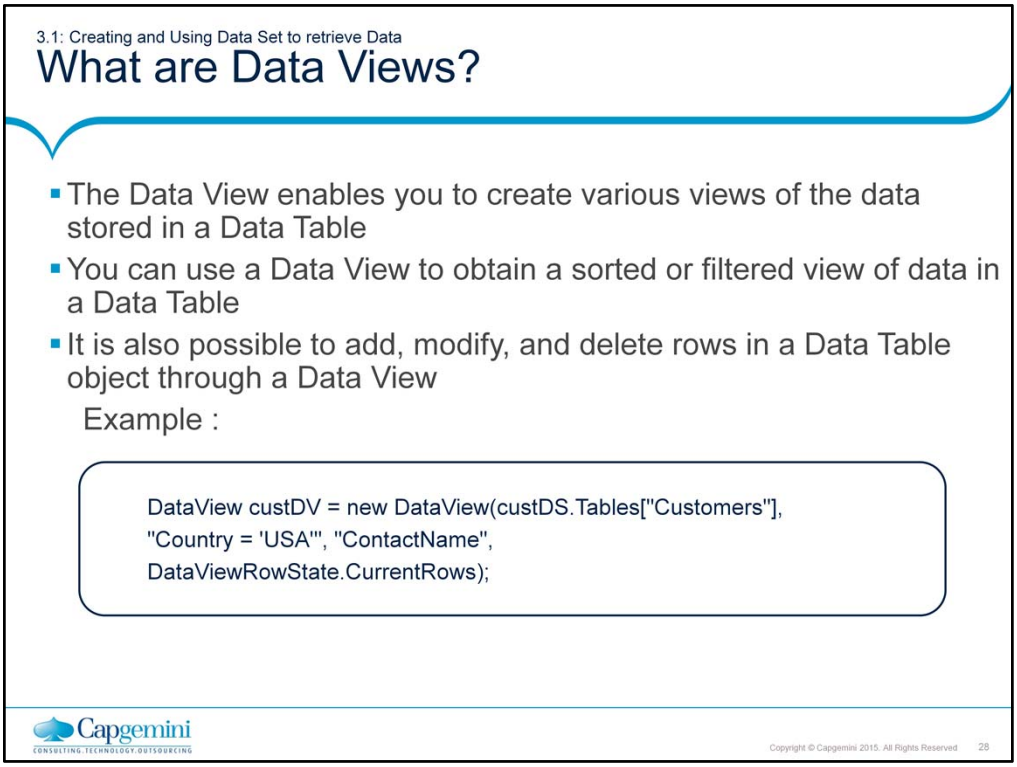
3.1: Creating and Using Data Set to retrieve Data

Demo

- Using the Data Relation object

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 27



What are DataViews?

A DataView enables you to create different views of the data stored in a DataTable, a capability that is often used in data-binding applications.

Using a DataView, you can expose the data in a table with different sort orders, and you can filter the data by row state or based on a filter expression

A DataView provides you with a dynamic view of a single set of data, much like a database view, to which you can apply different sorting and filtering criteria.
In some respects, a DataView is similar to a view in a database. However, a DataView is subject to several restrictions that do not apply for database views, including:

You cannot treat a DataView as a table.
You cannot use a DataView to provide a view of joined DataTable objects.
A DataView cannot exclude columns that exist in the source DataTable.

The following code example demonstrates how to create a DataView using the DataView constructor. A RowFilter, Sort column, and DataViewRowState are supplied along with the DataTable.

DataView custDV = new DataView(custDS.Tables["Customers"], "Country = 'USA'", "ContactName", DataViewRowState.CurrentRows);

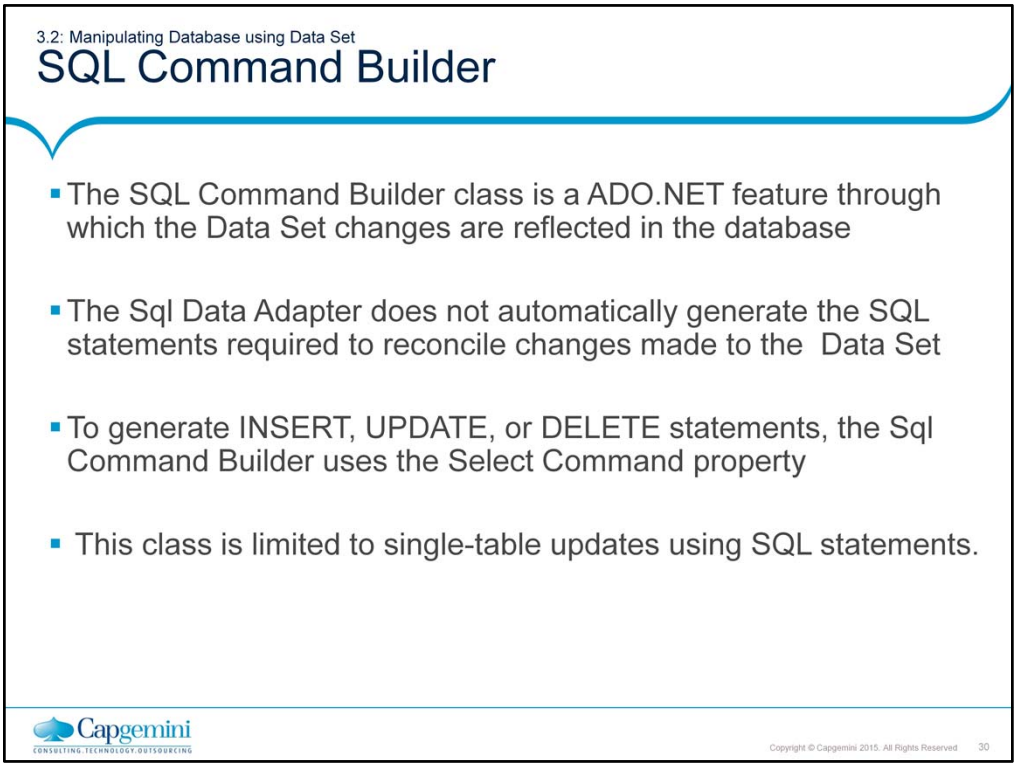
3.1: Creating and Using Data Set to retrieve Data

Demo

- Using the Data View object

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 29



SQLCommandBuilder

The SQLCommandBuilder class is a ADO.NET feature through which the DataSet changes are reflected in the database. When an instance of SQLCommandBuilder class is created it automatically generates Transact-SQL statements for the single table updates that occur. A CommandBuilder is managed provider-specific class that works on top of the data adapter object. The SQLCommandBuilder runs the SelectCommand to collect the required information about the tables and columns involved and then creates InsertCommand, DeleteCommand and UpdateCommand. The command builder class ensures that the specified data adapter can be successfully used to update the given data source. It uses some of properties defined for the SelectCommand like connection, commandtimeout and transaction. Whenever any of these properties are modified, you need to call the CommandBuilder's RefreshSchema method to change the structure of the generated commands for further updates.
The command builder is useful because it lets you update the data source withchanges made to the DataSet using very little code. It also lets you create update logic without understanding how to code the actual delete, insert, and update SQL statements. There are drawbacks, however, including slower performance because of thetime that it takes to request metadata and construct the updating logic, updates thatare limited to simple single-table scenarios, and a lack of support for stored procedures.

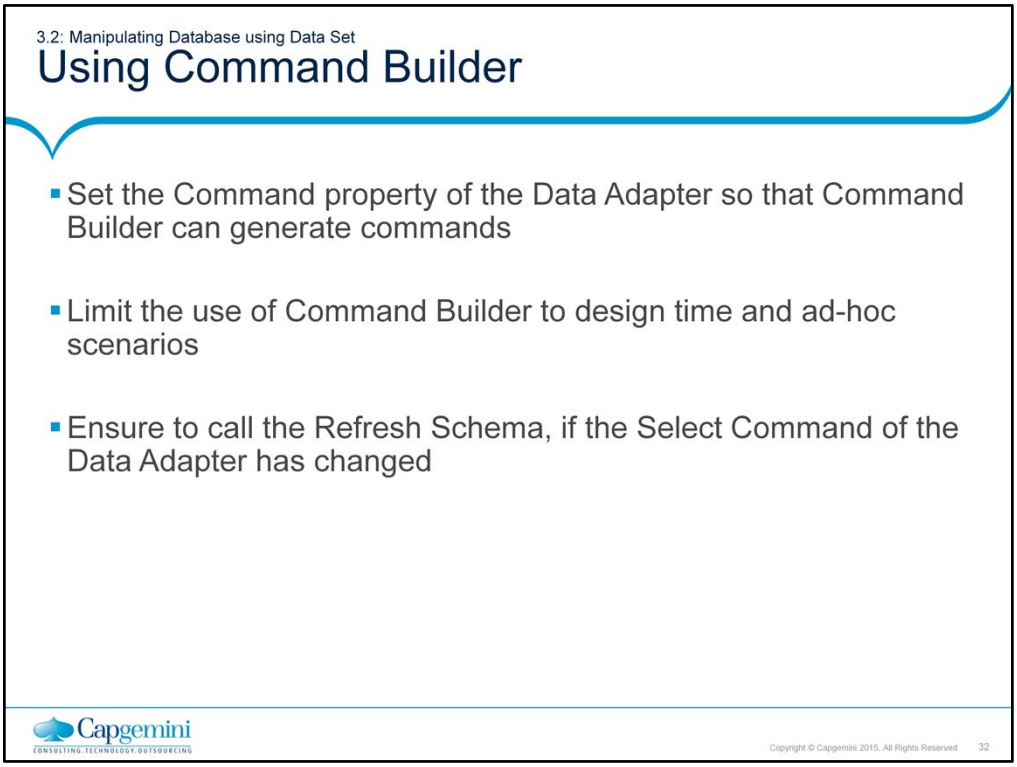
3.2: Manipulating Database using Data Set

Demo

- Using SQL Command Builder

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 31



Managing DataSet and Other Objects:

Using CommandBuilder:

As we already know, the CommandBuilder automatically generates the following properties of a DataAdapter based on the SelectCommand property of the DataAdpater:

InsertCommand
UpdateCommand
DeleteCommand

To use the CommandBuilder effectively:

Set the CommandProperty to null for the DataAdapter, only then CommandBuilder generates the commands. If you have explicitly set a CommandProperty then CommandBuilder does not override it.
Limit the use of CommandBuilder to design time or ad-hoc scenarios. The processing required to generate the DataAdapter command property actually hampers the performance. If the contents of Insert / Update / Delete are known then set them explicitly. Alternatively you can design stored procedures for the commands and configure the DataAdapter to use them.
CommandBuilder uses the SelectCommand property of DataAdapter to determine the values for other command properties. Call the RefreshSchema, if the SelectCommand of the DataAdapter had changed.

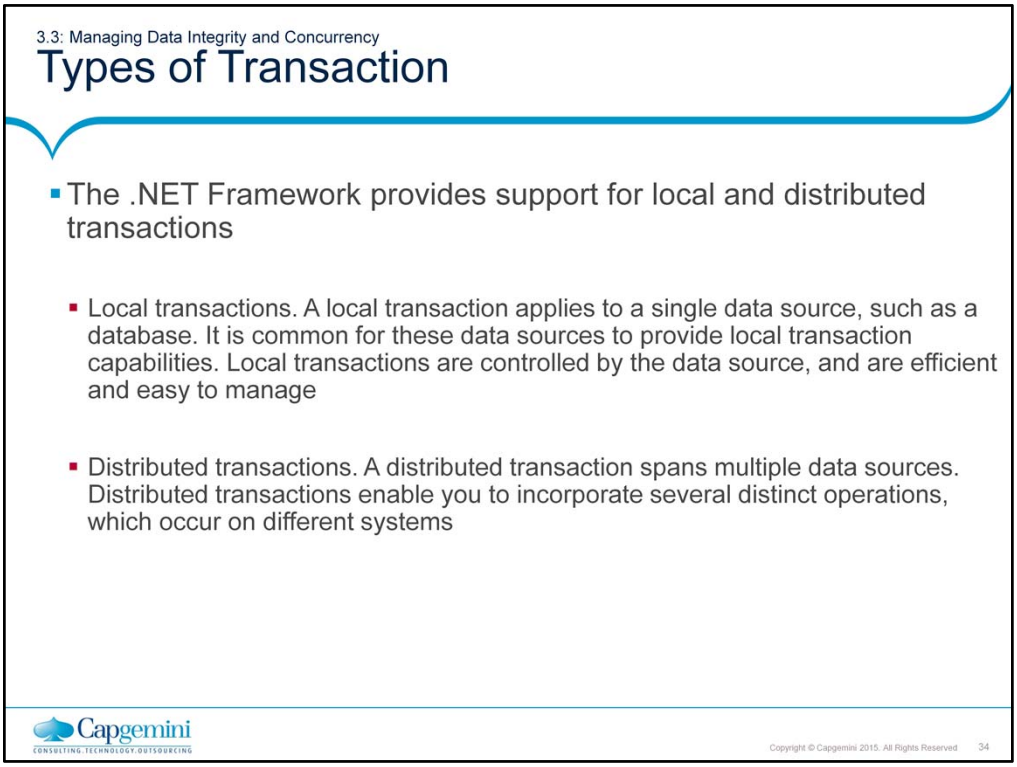
3.3: Managing Data Integrity and Concurrency

Maintaining Database Integrity - Transaction

- A transaction is a unit of work
- You use transactions to ensure the consistency and integrity of a database
- If a transaction is successful, all of the data modifications performed during the transaction are committed and made permanent
- If an error occurs during a transaction, you can roll back the transaction to undo the data modifications that occurred during the transaction

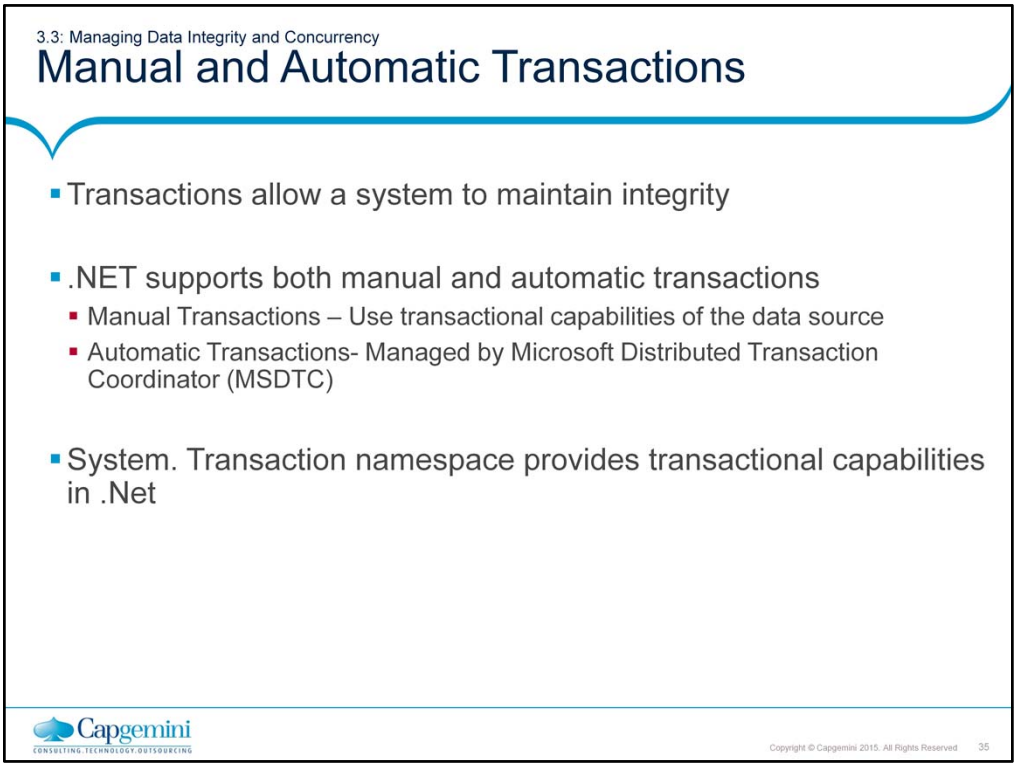
Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 33



Example of Distributed Transaction: Say you ordered something from flipcart.com and made the payment from your bank account. Your bank account server and flipcart server are two different computers in two different networks and data is updated on both the servers (Amount is being deducted from your bank account and added to the bank account of flipcart).

Example of Local Transaction: You book a Fixed Deposit using NetBanking.

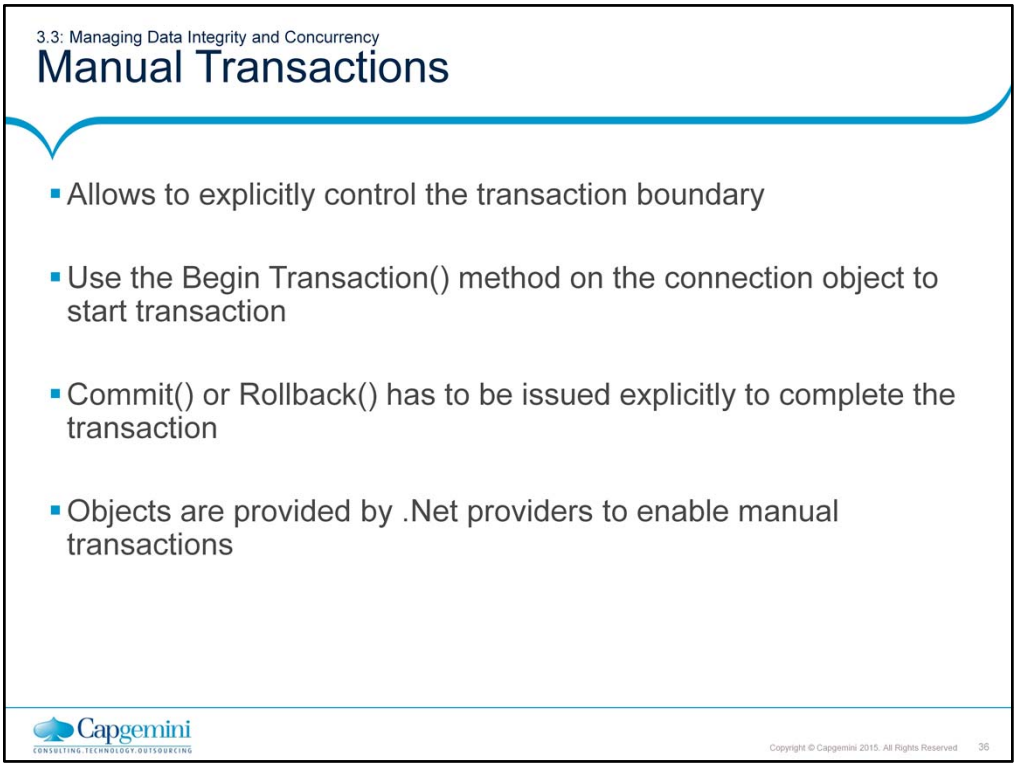


Maintaining Database Integrity:

Transactions allow a system to maintain integrity when interacting with multiple data sources. Both manual and automatic transactions are supported by .Net. The System.Transaction namespace provides with transactional capabilities in .Net.

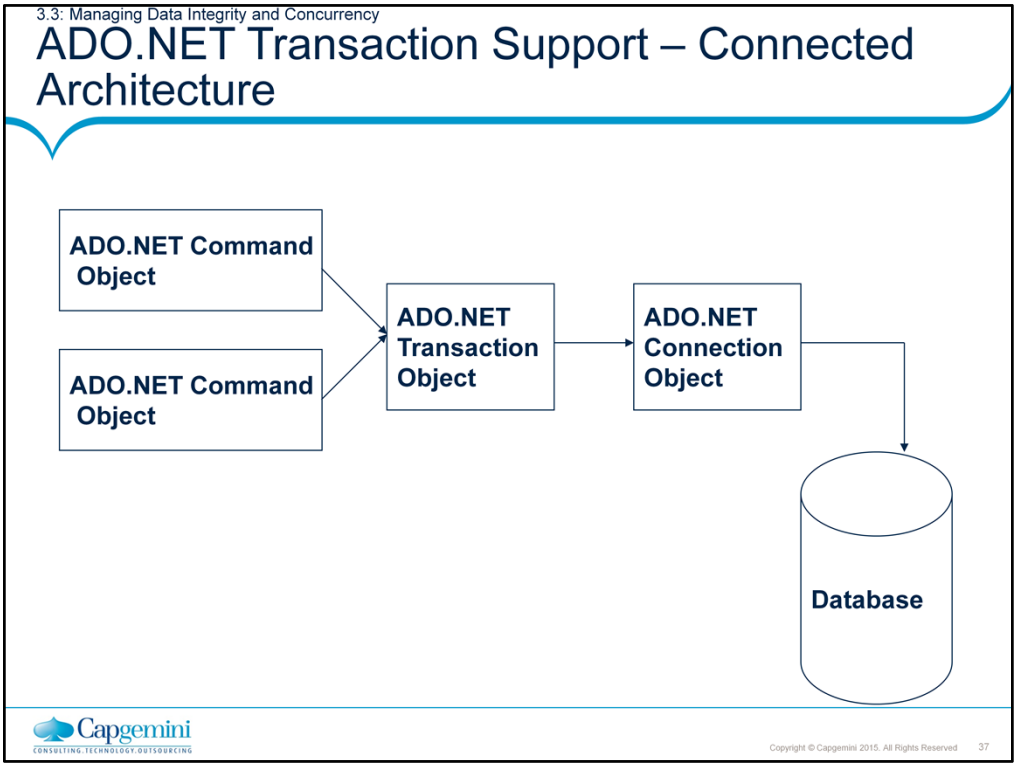
Manual Transactions: In this, a transaction object is associated with the data source and transactional capabilities of data source is used. One transaction can have multiple commands executed against the data source. Manual transactions can be controlled by using the SQL Commands and they are limited to carry out transactions against a single data source. Manual transactions are faster as compared to automatic transactions since they do not need interprocess communication with MSDTC.

Automatic Transactions: As against manual transactions, automatic transactions can span multiple data sources and they are comparatively slower than manual transactions.
(Note: Distributed Transactions are not in the scope of this course)
Whenever multiple users attempt to modify unlocked data concurrency problems occur. Locking data ensures database consistency by controlling how changes made to data within an uncommitted transaction can be used by concurrent transactions.



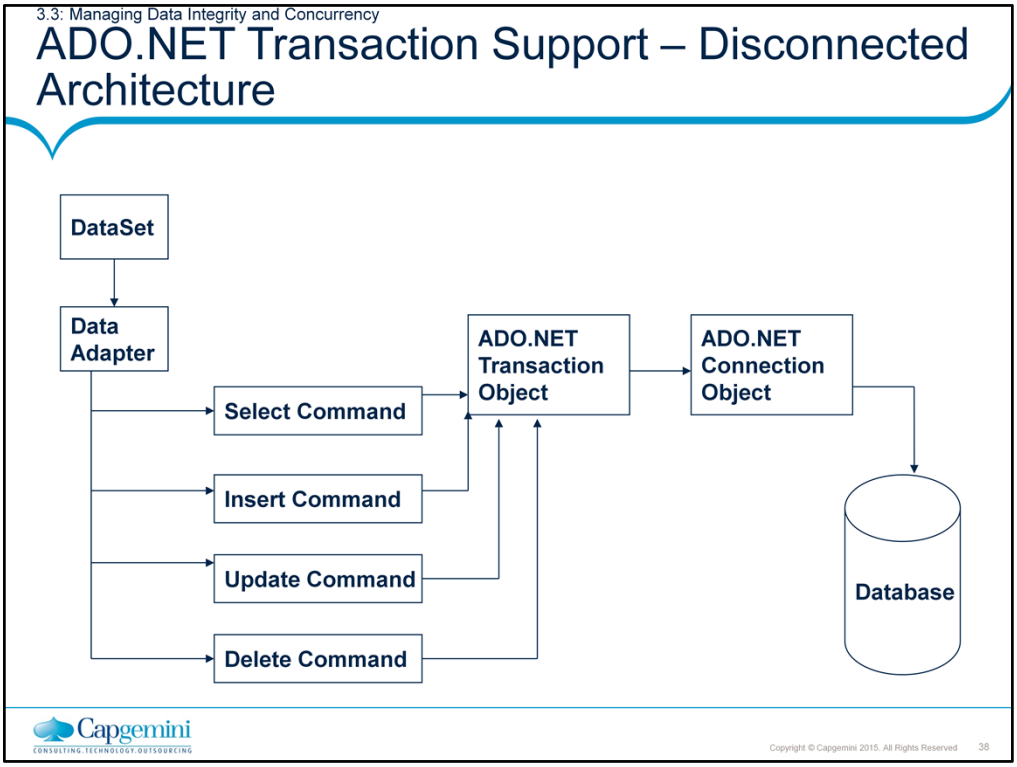
Manual Transactions:

Manual transactions allow control over the transaction boundary. You need to provide explicit instructions to start and end the transaction. Objects are provided by .Net providers to enable manual transactions. The Connection object gives a BeginTransaction() method that is used to start a transaction. The method returns a transaction object if successful, which can then be used to perform all subsequent actions associated with the transaction. The transaction object returned by the BeginTransaction() has to be associated with the Command object to execute the command within the transaction.
The transaction does not complete automatically, the Commit() or Rollback() methods have to be issued to complete the transaction. The Commit() method of the Transaction is used to commit the changes made to database through the transaction and Rollback() method of the transaction is used to undo the changes made to the database. In case the rollback is called after the commit then a InvalidOperationException() is raised.



In connected mode, the typical sequence of operations in a transaction will be as follows:
1. Open a database connection.
2. Begin a transaction.
3. Fire queries directly against the connection via the command object.
4. Commit or roll back the transaction.
5. Close the connection.

Implementing transactions in connected mode is relatively simple, as we have everything happening live; however, in disconnected mode, while updating the data back into the database, some care should be taken to account for concurrency issues.



In disconnected mode, generally, data is fetched first, usually one or more tables into a DataSet object, the connection with the data source is closed, the data is manipulated as required, and then the data is updated back into the database. In this mode, the typical sequence of operations will be as follows:

1. Open a database connection.
2. Fetch the required data in a DataSet object.
3. Close the database connection.
4. Manipulate the data in the DataSet object.
5. Again, open a connection with the database.
6. Start a transaction.
7. Assign the transaction object to the relevant commands on the data adapter.
8. Update the database with changes from the DataSet.
9. Close the connection.

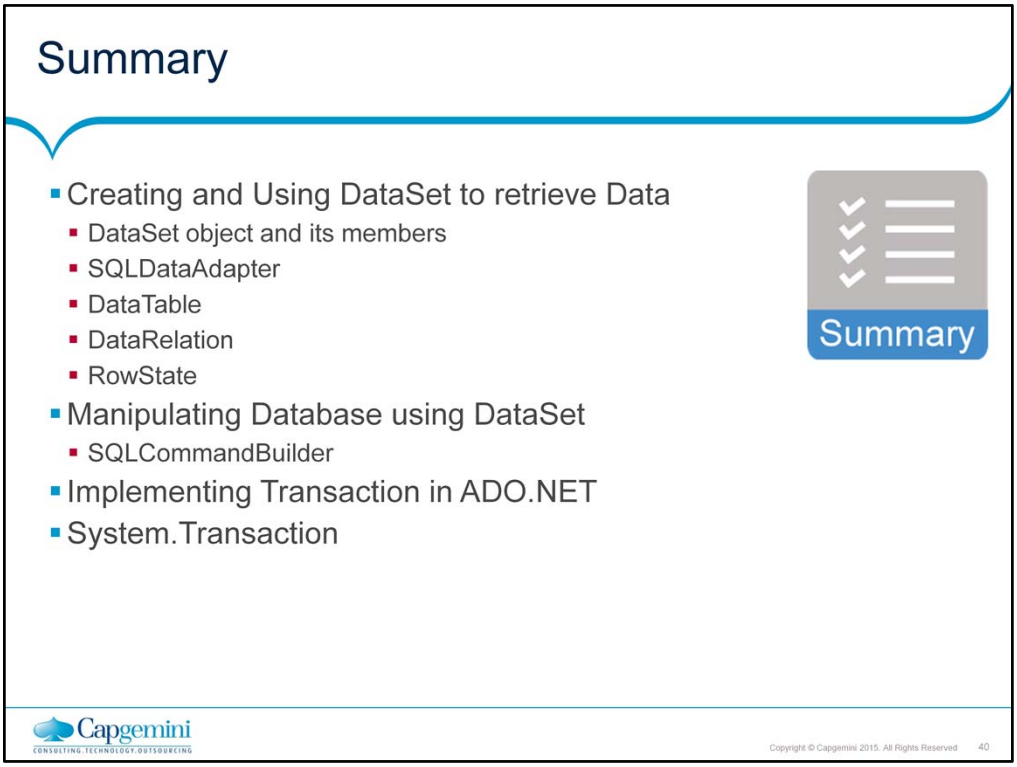
3.3: Managing Data Integrity and Concurrency

Demo

- Transactions

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 39



Add the notes here.

Review Question

- Question 1: Data Set has the following properties
  - Option 1:Memory Resident data representation
  - Option 2:Disconnected from Data Source
  - Option 3:Both the above
- Question 2: Data Table is created within the
  - Option 1:DataAdpater
  - Option 2:DataSet
  - Option 3:Data Relation
- Question 3: Row State property value depends on which operation has been performed on the row
  - True/False

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 41

Add the notes here.

ADO.NET 4.5

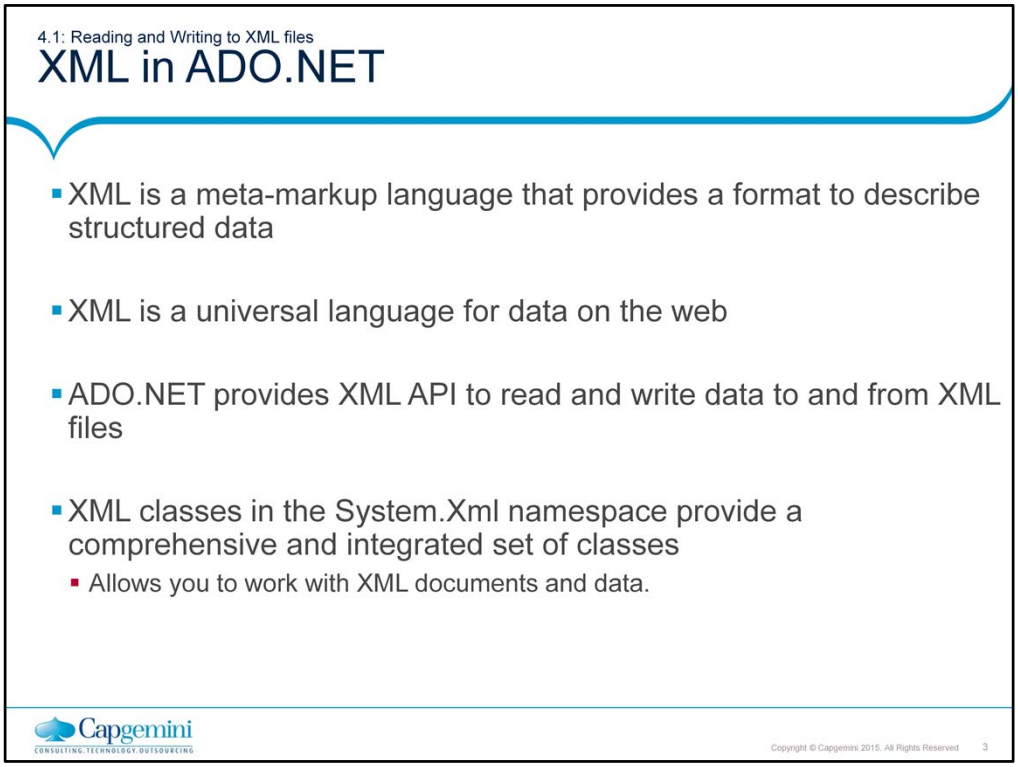
Lesson 4: XML support in ADO.NET

Lesson Objectives

- In this lesson, you will learn:
  - System.XML Namespace
    - Reading and Writing to XML files
  - XML and Relational Data

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

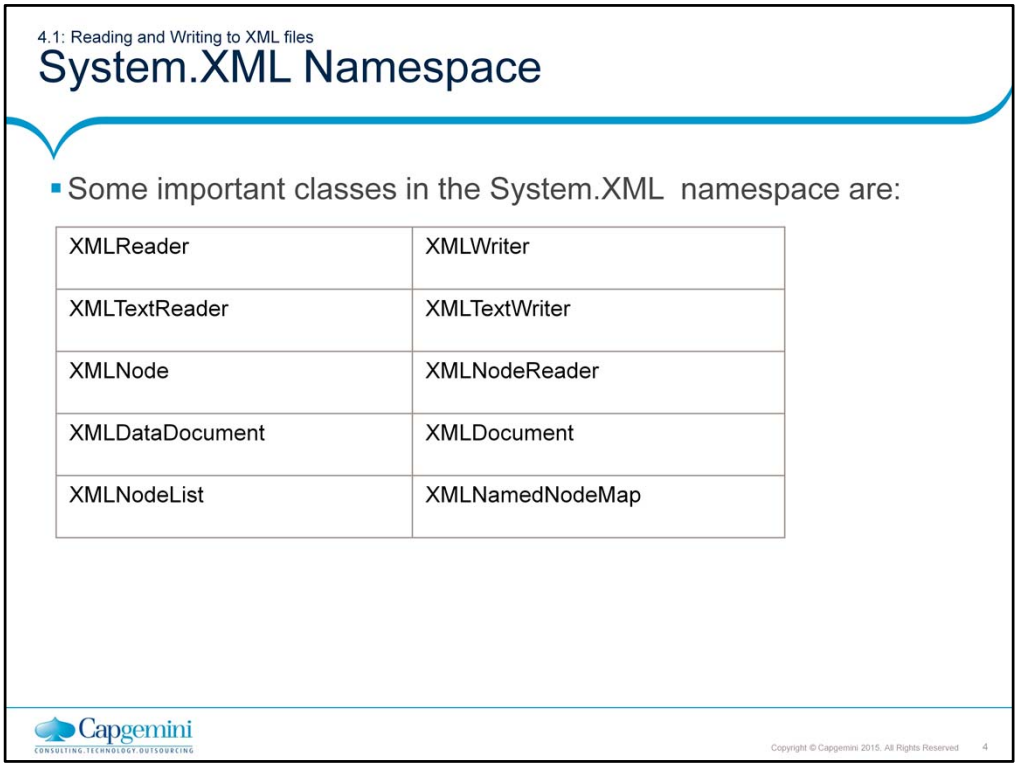
 2



Extensible Markup Language (XML) is a meta-markup language that provides a format for describing structured data. XML enables a new generation of Web-based data viewing and manipulation applications. XML is the universal language for data on the Web. XML gives developers the power to deliver structured data from a wide variety of applications to the desktop for local computation and presentation. It is used extensively in applications that are written by using general-purpose languages like C# or VB.NET.
It is mostly used for the following:

to exchange data between applications,
to store configuration information,
to persist temporary data,
as a base for generating web pages or reports, and
for many other things.

The XML API provided through System.Xml namespace provide a comprehensive and integrated set of classes, allowing you to work with XML documents and data. This namespace has a comprehensive set of XML classes for parsing, validation, and manipulation of XML data using readers, writers, and World Wide Web Consortium (W3C) DOM-compliant components. It also covers XML Path Language (XPath) queries and Extensible Stylesheet Language Transformations (XSLT). You should note that the XML namespace allows you to get similar results in a number of different ways .



Support for processing XML is provided by the classes in the System.Xml namespace in .NET. Some of the important classes are:

XMLReader - An abstract reader class that provides fast, non-cached XML data. XmlReader is forward-only.

XMLWriter - Represents an abstract writer that provides a fast, non-cached, forward-only means of generating streams or files containing XML data that conforms to the W3C Extensible Markup Language (XML) 1.0 and the Namespaces in XML recommendations.

XMLTextReader – This class extends XmlReader and conforms to the W3C Extensible Markup Language (XML) 1.0 and the Namespaces in XML recommendations. XmlTextReader provides the following functionality:

Enforces the rules of well-formed XML.
XmlTextReader does not provide data validation.
Checks that DocumentType nodes are well-formed. XmlTextReader checks the DTD for well-formedness, but does not validate using the DTD.
Does not expand default attributes.

XMLTextWriter - This class extends the XMLWriter. Represents a writer that provides a fast, non-cached, forward-only way of generating streams or files containing XML data that conforms to the W3C Extensible Markup Language (XML) 1.0 and the Namespaces in XML recommendations. XmlTextWriter does not check for the following:

- **Invalid characters** in attribute and element names.
- **Unicode characters** that do not fit the specified encoding. If the Unicode characters do not fit the specified encoding, the XmlTextWriter does not escape the Unicode characters into character entities.
- **Duplicate attributes** Characters in the DOCTYPE public identifier or system identifier.

- **XMLNode** – An abstract class that represents a single node in an XML document. XmlNode is the base class in the .NET implementation of the DOM.

- **XMLNodeReader** – This class represents a reader that provides fast, non-cached forward only access to XML data in an XmlNode and extends the XMLReader**.** The XmlNodeReader has the ability to read an XML DOM subtree. This class does not support DTD or schema validation.

- **XMLDocument** – Represents an XML document and it extends XMLNode. It provides a tree representation in memory of an XML document, enabling navigation and editing.

- **XMLDataDocument** – This class extends the XMLDocument. Allows structured data to be stored, retrieved, and manipulated through a relational DataSet. Allows the mixing of XML and relational data in the same view.

- **XMLNodeList** – To represents an ordered collection of nodes you can use XMLNodeList. The XmlNodeList collection reflects changes to the children of the node object that it was created from immediately in the nodes returned by the XmlNodeList properties and methods. XmlNodeList supports iteration and indexed access.

  - **XmlNodeList** is returned by the following properties and methods.
  - **XmlNode.ChildNodes** Returns a XmlNodeList containing all the children of the node.
  - **XmlNode.SelectNodes** XmlNodeList is returned containing a collection of nodes matching the XPath query.
  - **GetElementsByTagName** Returns XmlNodeList containing a list of all descendant elements that match the specified name. This method is available in both the XmlDocument and XmlElement classes.

- **XMLNamedNodeMap** – This class represents a collection of nodes that can be accessed by name or index. XmlNamedNodeMap is returned by the following three properties.
  - **XmlElement.Attributes** Returns XmlAttributeCollection, a class which inherits from XmlNamedNodeMap.
  - **XmlDocumentType.Entities** Returns an XmlNamedNodeMap containing XmlEntity objects. The XmlNamedNodeMap is read-only.
  - **XmlDocumentType.Notations** Returns an XmlNamedNodeMap containing XmlNotation objects. The XmlNamedNodeMap is read-only.

4.1: Reading and Writing to XML files

Demo

- Using the XML Text Reader

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 6

4.1: Reading and Writing to XML files

Demo

- Using the XML Document

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 7

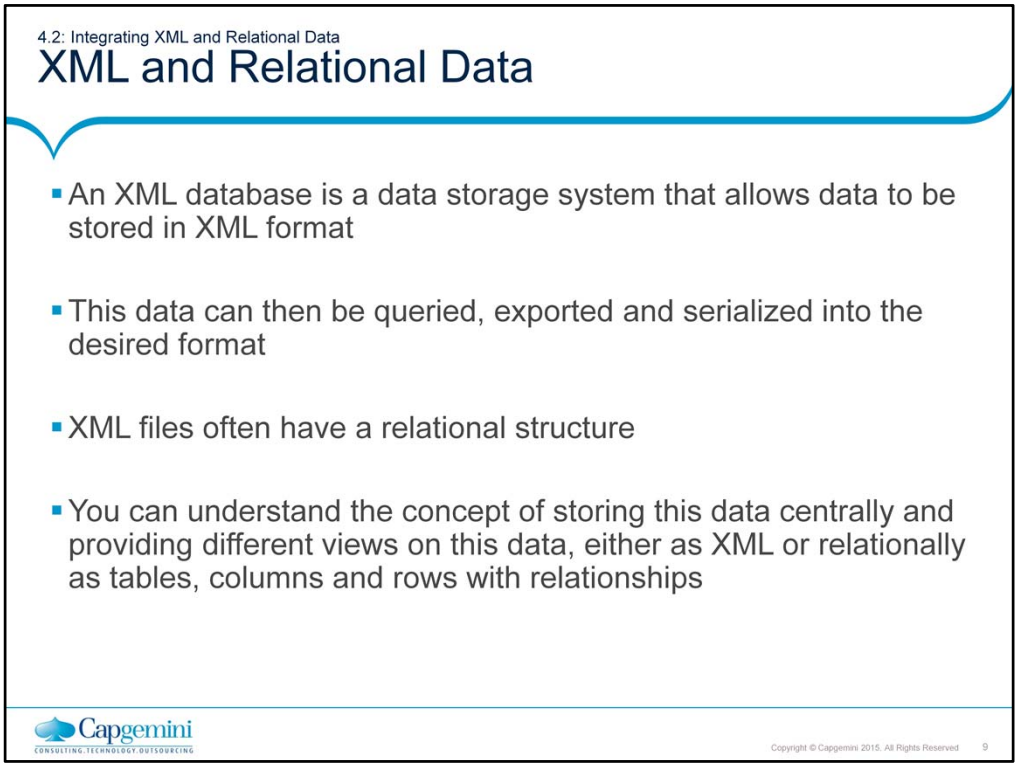
4.1: Reading and Writing to XML files

Comparison – XML Reader & XML document

- Use XML Document when:
  - You wish to perform operation such as:
    - Insert, Update and Delete
  - Memory is not a constraint.
- Use XML Reader when:
  - You wish to read data in a forward-only manner.
  - When memory is constraint.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

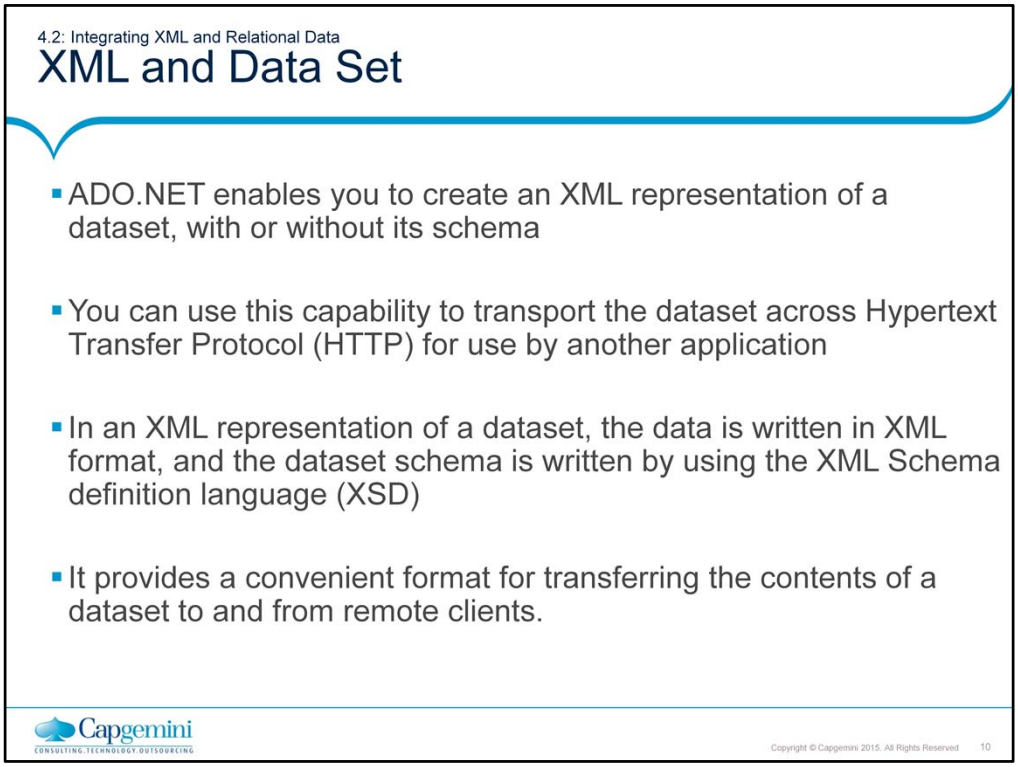
 8



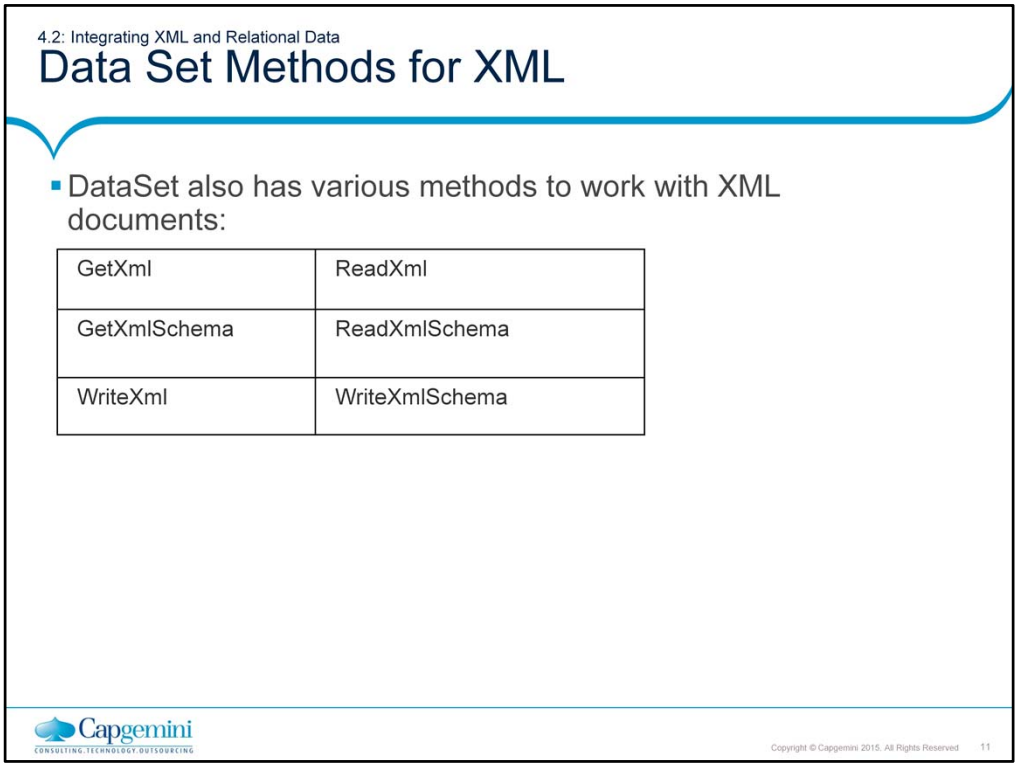
XML often has a relational structure (the fictional books in a bookstore for example), as well as being structured. This set of topics cover the concept of storing this data centrally and providing different views on this data, either as XML or relationally as tables, columns and rows with relationships. This disconnected store of data which, for example, could represent a business object in the middle tier that enforces business rules, could provide its data as XML to a browser via Extensible Stylesheet Language Transformations (XSLT), across the Internet to another Web site, or to a local application via the relational tables. However, the data gets amended, you can update that data to a database on a transaction basis.

A DataSet represents an in-memory cache of data as a collection of tables and relationships between those tables. It is, in effect, a locally cached database. This provides a disconnected cache of data, like a message, that enables dealing with chunks of data. The DataSet has no knowledge of where the data came from. It may have come from a file, a database connection, or from a stream. A DataSet provides a relational view onto this stored data.

The XmlDataDocument provides XML APIs for accessing this in-memory cache of data, as well as supporting reading and writing XML. The XmlDataDocument is a DataSet-aware object. Creation of an XmlDataDocument implicitly creates a DataSet (accessed as a property) that provides a relational view onto the data XML data. This symbiotic relationship between these two objects provides a powerful technique for accessing data either relationally or as XML, irrespective of the mechanism by which the data was sourced.

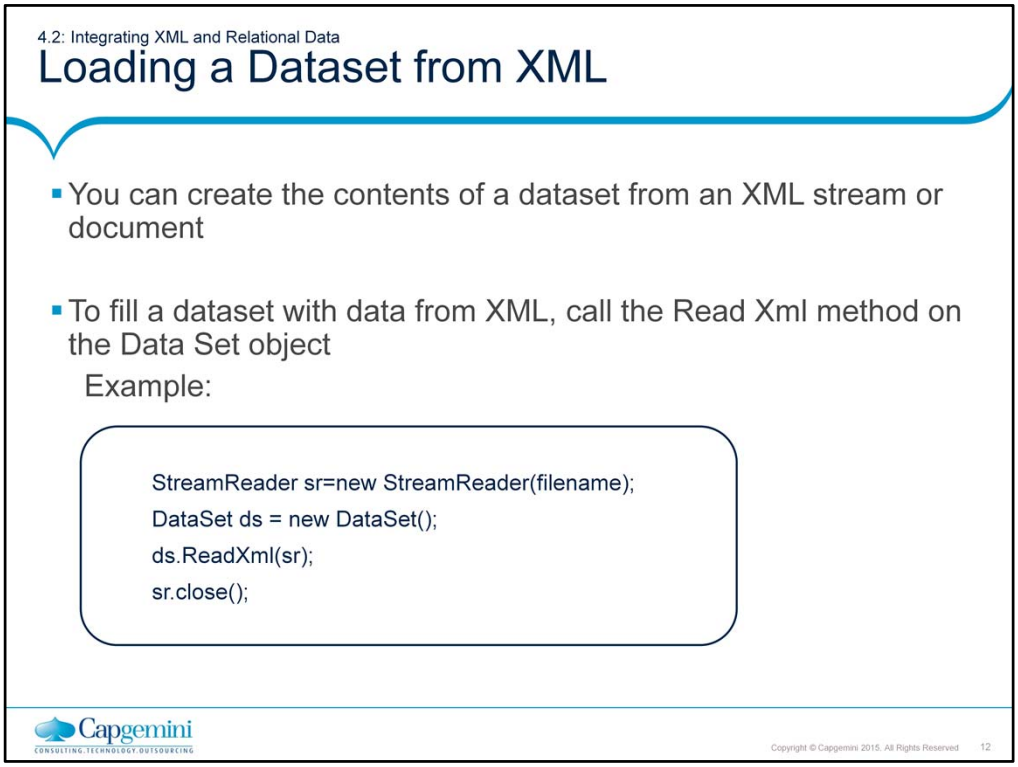


With ADO.NET you can fill a DataSet from an XML stream or document. XmlDataDocument is a DataSet-aware object. Relationship between these two objects provides a powerful technique to access data either relationally or as XML, irrespective of the mechanism by which the data is sourced.
You can use the XML stream or document to supply to the DataSet either data, schema information, or both. The information supplied from the XML stream or document can be combined with existing data or schema information already present in the DataSet.
ADO.NET also allows you to create an XML representation of a DataSet, with or without its schema, in order to transport the DataSet across HTTP for use by another application or XML-enabled platform. In an XML representation of a DataSet, the data is written in XML and the schema, if it is included inline in the representation, is written using the XML Schema definition language (XSD). XML and XML Schema provide a convenient format for transferring the contents of a DataSet to and from remote clients.



The DataSet is silently rebuilt as a binary and promptly usable object. The same serialization facilities are available to applications through a bunch of methods, a pair of which clearly stands out. The methods to work with XML Document offered by Dataset are:

GetXml: Returns a string value of the XML representation of the data stored in the DataSet.
GetXmlSchema: Returns the XML Schema for the XML representation of the data stored in the DataSet.
ReadXml: Populates a DataSet object with the specified XML data read from a stream or a file.
ReadXmlSchema: Loads the specified XML schema information into the current DataSet object.
WriteXml: Writes the XML data, and optionally the schema, that represents the DataSet. Can write to a stream or a file.
WriteXmlSchema: Writes the string being the XML schema information for the DataSet .Can write to a stream or a file.



The contents of an ADO.NET DataSet can be created from an XML stream or document. With the .NET Framework you have great flexibility over what information is loaded from XML, and how the schema or relational structure of the DataSet is created.

To fill a DataSet with data from XML, use the ReadXml method of the DataSet object. The ReadXml method will read from a file, a stream, or an XmlReader, and takes as arguments the source of the XML plus an optional XmlReadMode argument. The ReadXml method reads the contents of the XML stream or document and loads the DataSet with data. The method creates the relational schema for the DataSet depending on the read mode specified, and whether or not a schema already exists in the DataSet. The following code snippet illustrates the typical code you would use to load a DataSet from XML:

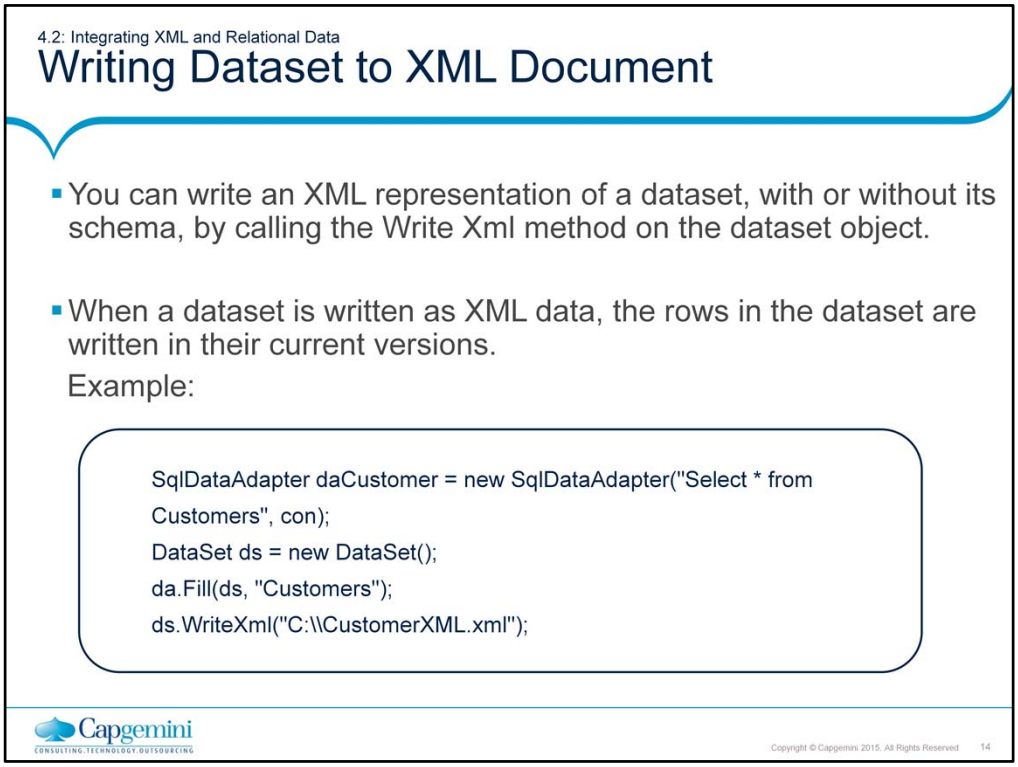
```
StreamReader sr=new StreamReader(filename);
DataSet ds = new DataSet();
ds.ReadXml(sr);
sr.close();
```

When loading the contents of XML sources into a DataSet, ReadXml does not merge rows whose primary key information match. To merge an existing DataSet with one loaded from XML, you first have to create a new DataSet, and then merge the two using the Merge method. During the merging, the rows that get overwritten are those with matching primary keys. An alternate way to merge existing DataSet objects with contents read from XML is through the DiffGram format.

Let us take a look at the various read modes the ReadXml supports:

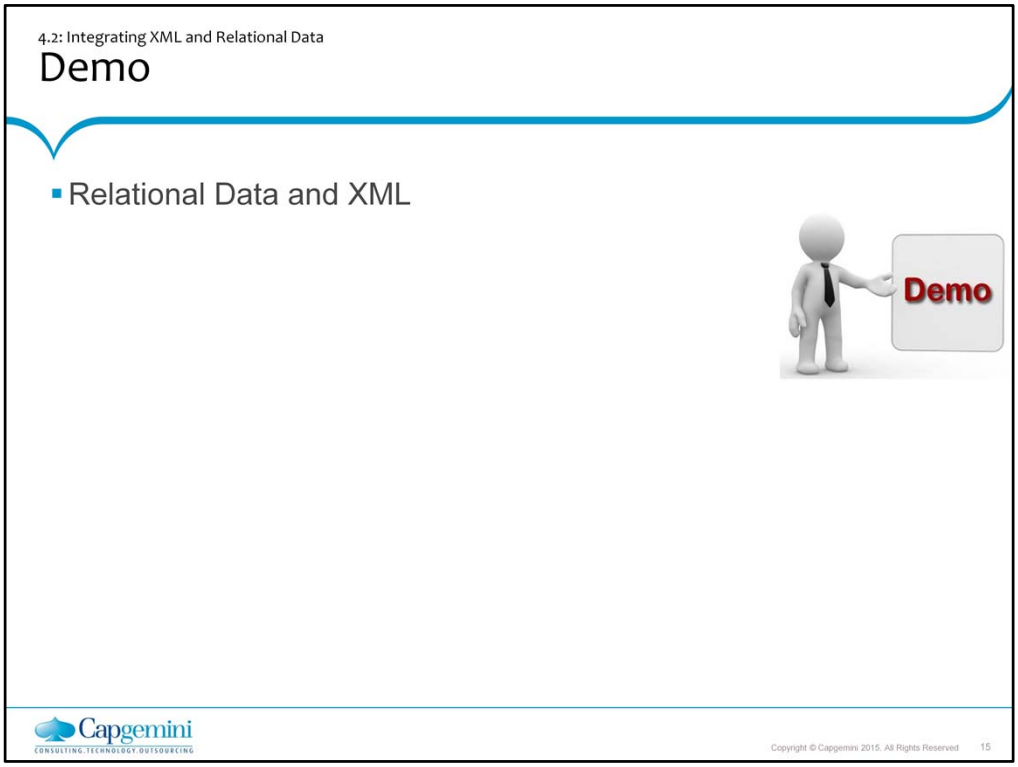
- **Auto**: This is the default. It examines the XML and chooses the most suitable option in the following order:
    - If the XML is a DiffGram, DiffGram is used.
    - If the DataSet contains a schema or the XML contains an inline schema, ReadSchema is used.
    - If the DataSet does not contain a schema and the XML does not contain an inline schema, InferSchema is used.

- **ReadSchema**: Reads any inline schema and loads the data and schema. If the DataSet already contains a schema, new tables are added from the inline schema to the existing schema in the DataSet. If the DataSet does not contain a schema, and there is no inline schema and no data is read.
- **IgnoreSchema**: Ignores any inline schema and loads the data into the existing DataSet schema. Any data that does not match the existing schema is discarded. If no schema exists in the DataSet, no data is loaded.
- **InferSchema**: Ignores any inline schema and infers the schema per the structure of the XML data, then loads the data.
- **DiffGram**: Reads a DiffGram and adds the data to the current schema. DiffGram merges new rows with existing rows where the unique identifier values match.
- **Fragment**: It continues reading multiple XML fragments until the end of the stream is reached. Fragments that match the DataSet schema are appended to the appropriate tables. The fragments that do not match the DataSet schema are discarded.



In ADO.NET you can write an XML representation of a DataSet, with or without its schema. If schema information is included inline with the XML, it is written using the XML Schema definition language (XSD). The schema contains the table definitions of the DataSet as well as the relation and constraint definitions.
When a DataSet is written as XML data, the rows in the DataSet are written in their current versions. However, the DataSet can also be written as a DiffGram so that both the current and the original values of the rows is included.
The XML representation of the DataSet can be written to a file, a stream, an XmlWriter, or a string. These choices provide great flexibility for how you transport the XML representation of the DataSet. To obtain the XML representation of the DataSet as a string, use the GetXml method.
To write a DataSet to a file, stream, or XmlWriter, use the WriteXml method. The first parameter you pass to WriteXml is the destination of the XML output. For example, pass a string containing a file name, a System.IO.TextWriter object, and so on. You can pass an optional second parameter of an XmlWriteMode to specify how the XML output is to be written.

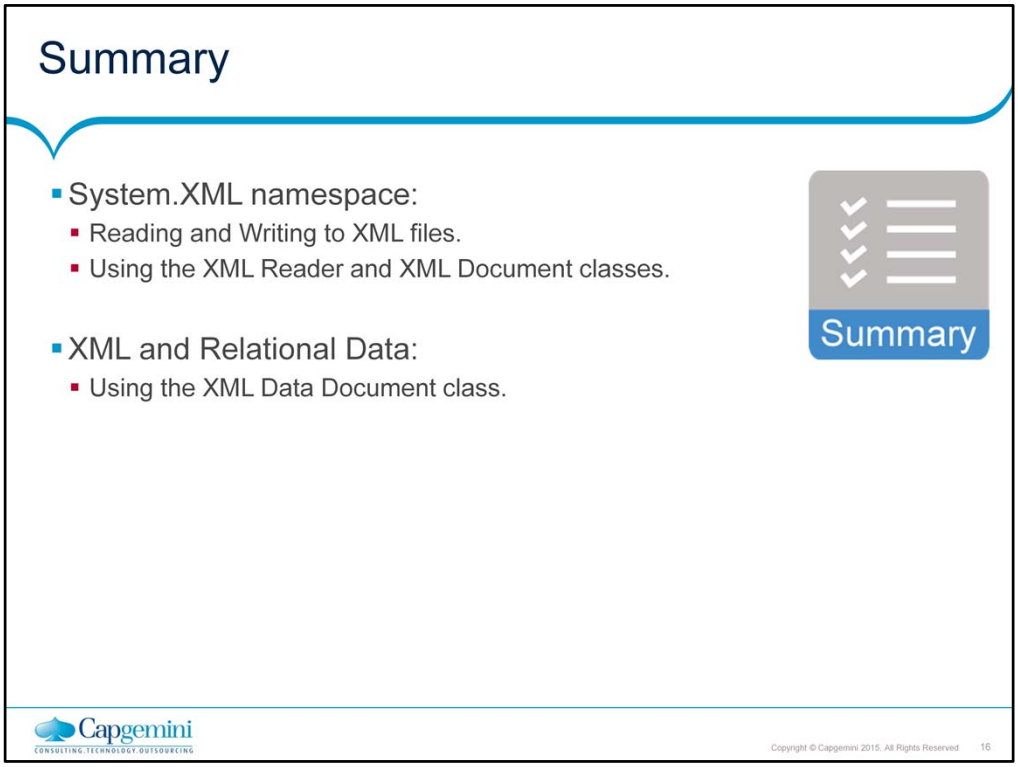
Write modes are as follows:
IgnoreSchema: Writes the current contents of the DataSet as XML data, without an XML Schema.
WriteSchema: Writes current contents of the DataSet as XML data with the relational structure, as inline XML Schema.
DiffGram: Writes the entire DataSet as a DiffGram, including original and current values.



When writing an XML representation of a DataSet that contains DataRelation objects, you will most likely want the resulting XML to have the child rows of each relation nested within their related parent elements. To accomplish this, set the Nested property of the DataRelation to true when you add the DataRelation to the DataSet.

Following code snippet shows how to use WriteXml:

```
System.IO.StreamWriter xmlSW = new
System.IO.StreamWriter("Customers.xml");
custDS.WriteXml(xmlSW, XmlWriteMode.WriteSchema);
xmlSW.Close();
```



Add the notes here.

Review Questions

- Question 1: Which XML class from System.XML namespace is Data Set aware?
  - Option 1: XML Document
  - Option 2: XML Data Document
  - Option 3: X Document
- Question 2: XML Reader reads data in a forward-only manner,
  - Option 1: True
  - Option 2: False

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 17

Add the notes here.

Review Questions

- Question 3: The _____ class allows you to directly go to a particular element without worrying about the root element.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Add the notes here.

ADO.NET 4.5

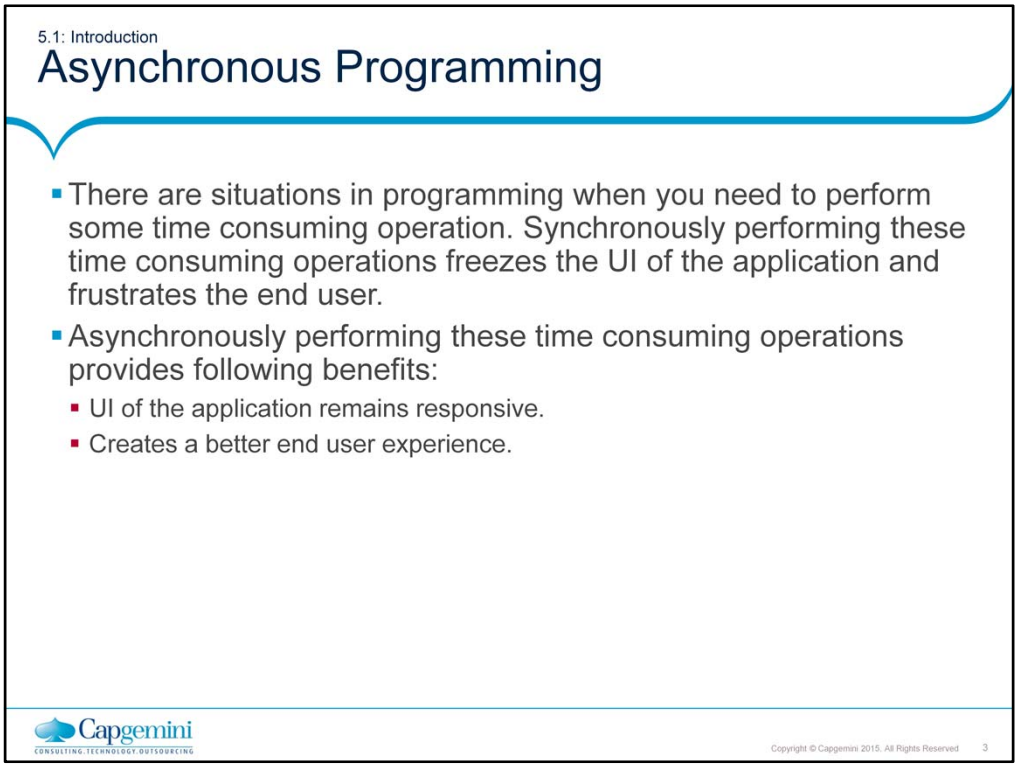
Lesson 5: Asynchronous Data Access

Lesson Objectives

- In this lesson, you will learn about:
  - Asynchronous Data Access in ADO.NET 4.5
  - Sql Credential

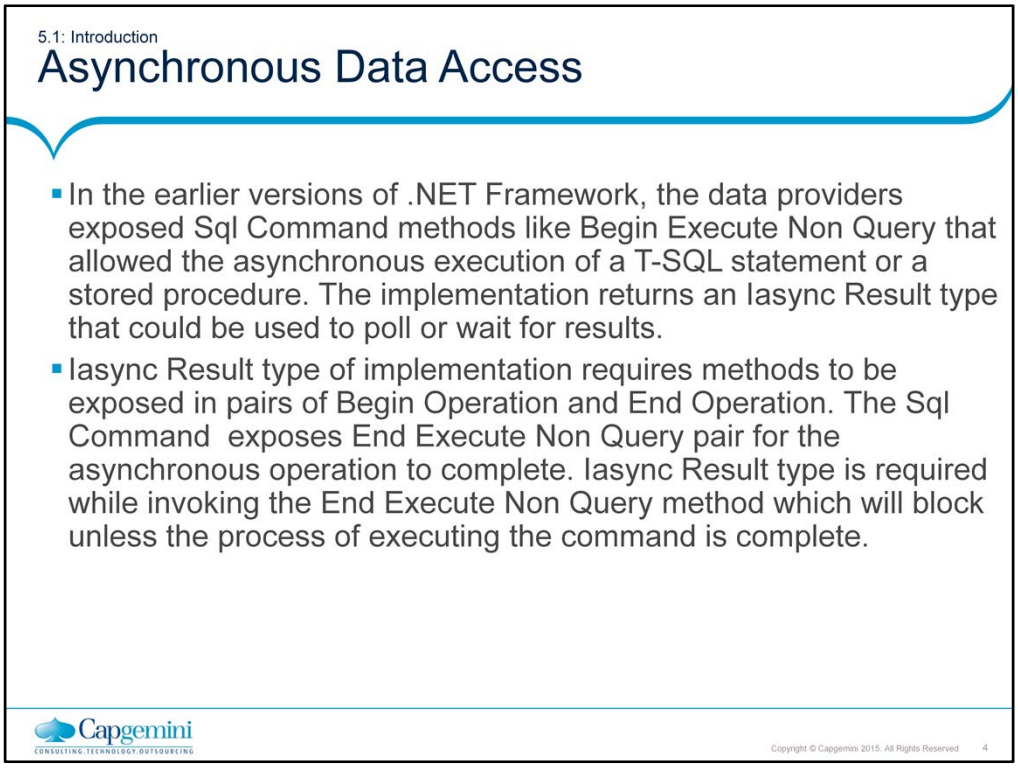
Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 2



With asynchronous programming new operation can be started and without waiting for this new operation to complete user can continue his work.
E.g.- When you are filling up a registration form, you enter your email address and that email address is verified for its validity. If you do this synchronously, then user won't be able to key in remaining information in the registration form unless email address is validated.

Asynchronous Programming can be used in following situations:
For performing some time consuming database operations
For performing some time consuming IO operation (Read/Write operation on a file).




```
using (var connection = new SqlConnection("..."))
{
try
{
var command = new SqlCommand(commandText, connection);
connection.Open();
var callBack = new AsyncCallback(CallBack);
var result = command.BeginExecuteNonQuery(callBack, command);
while (!result.IsCompleted)
{
//TODO: Continue to perform your other operations here
}
}
catch (SqlException)
{
//Log Exception
}

catch (Exception)
{
//Log Exception
}
}
```

You will notice that the main thread is not blocked, and you can continue to see the results inside the while loop
that checks for the IAsyncResult IsCompleted property. A simple callback handler will look like the following:

```
private static void CallBack(IAsyncResult result)
{
try
{
var command = (SqlCommand)result.AsyncState;
command.EndExecuteNonQuery(result);
}
catch (Exception)
{
//Log Exception
}
}
```

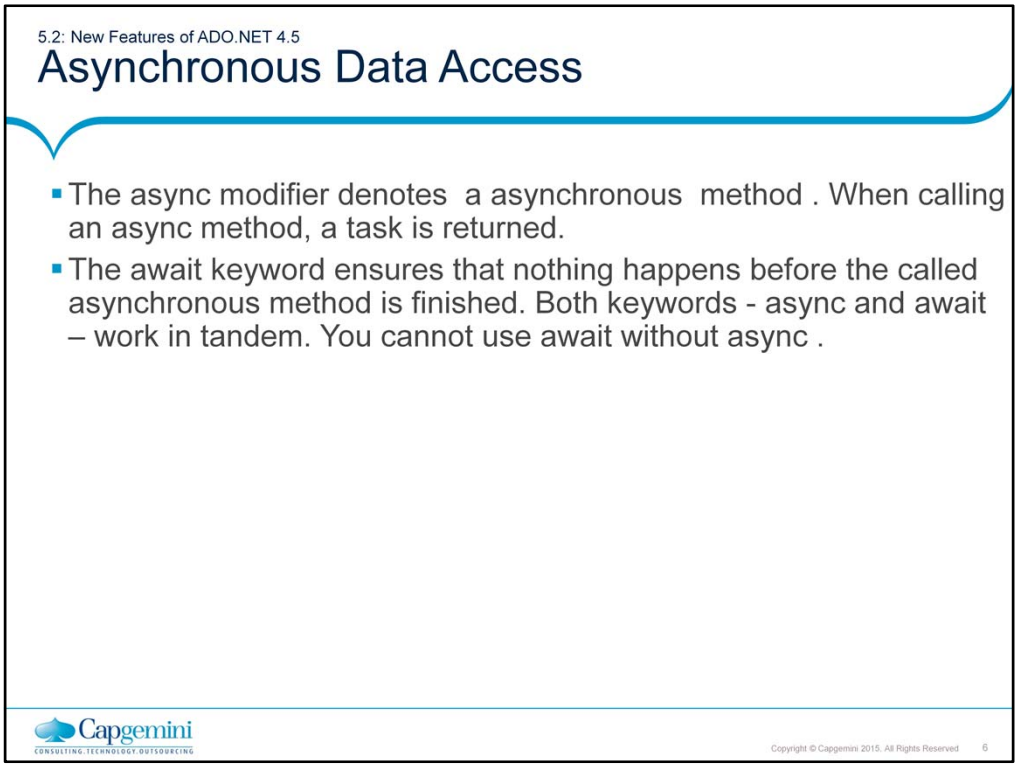
5.2: New Features of ADO.NET 4.5

Asynchronous Data Access

- Till .NET Framework 3.5 you need to specify a callback method that will be called to do some processing after asynchronous operation finishes.
- ADO.NET 4.5 introduces an overly simplified async programming model where you can perform asynchronous calls without using callbacks. The async and await modifiers in .NET Framework 4.5 are used for this purpose.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 5



For a simple asynchronous execution of the command, this is lot of code. You will find out how this is simplified in .NET Framework 4.5.

The ADO.NET Async programming model in .NET Framework 4.5 exposes an equivalent asynchronous method for every synchronous method exposed by the data providers. Continuing with the previous example, the ExecuteNonQuery method of SqlCommand class has an equivalent asynchronous method named ExecuteNonQueryAsync. It accepts a CancellationToken as a parameter allowing the operation to be aborted before the command timeout elapses.

Now let's take a look at how the programming model significantly reduces the lines of code you have to write for the same operation.

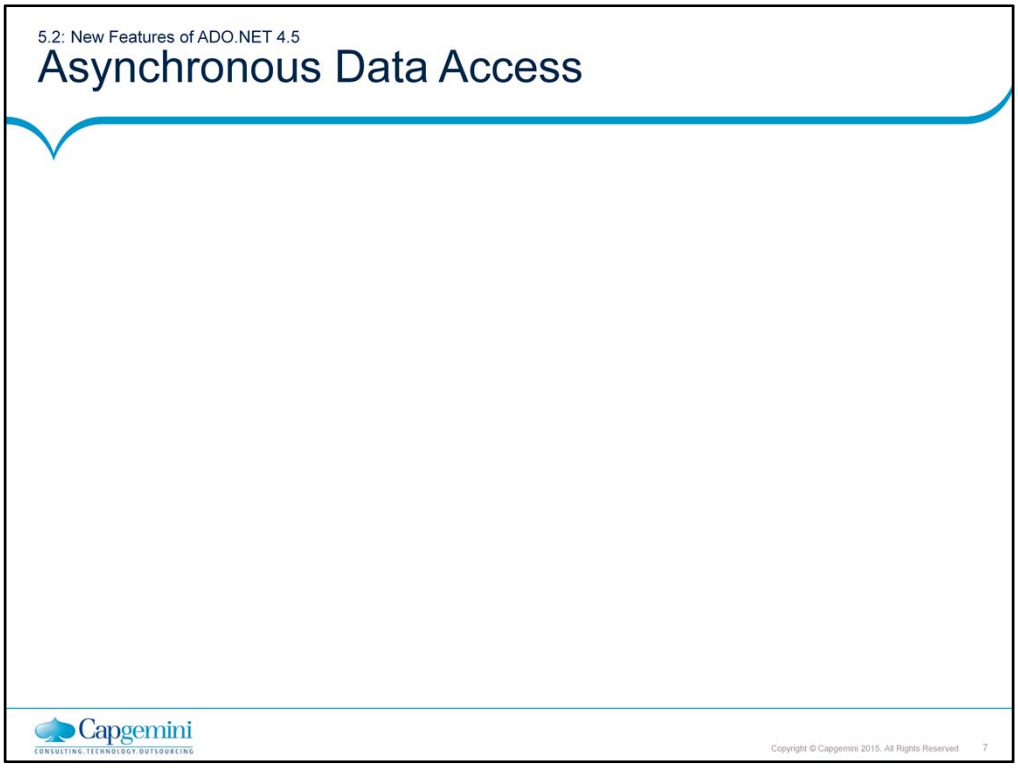
```
private static async Task<int> ExecuteCommandAsync(SqlConnection connection, SqlCommand command)
{
await connection.OpenAsync();
await command.ExecuteNonQueryAsync();
return 1;
}
```

A caller method can pass the connection and command instances to call this method and execute the T-SQL statement or procedure.

```
using (var connection = new SqlConnection("..."))
{
try
{
var command = new SqlCommand(commandText, connection);
int result = ExecuteCommandAsync(connection, command).Result;
}
catch (SqlException)
{
//Log Exception
}
catch (Exception)
{
//Log Exception
}
}
```

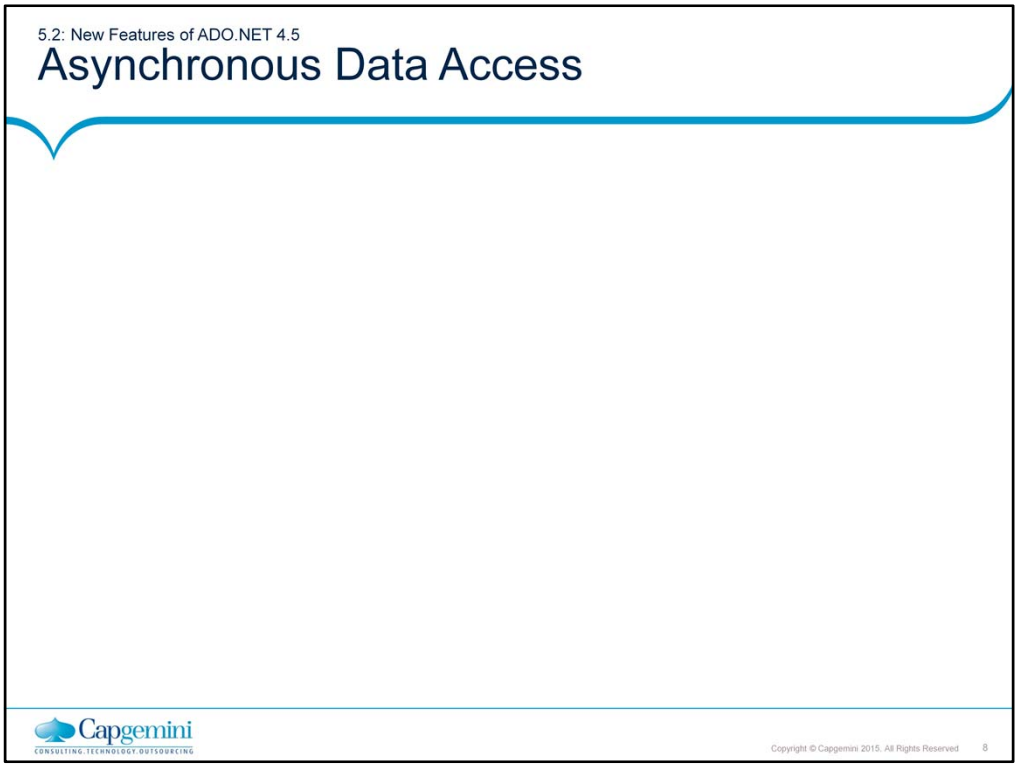
The async programming model is extremely robust, and it allows you to execute a task-based chain of commands. You could potentially use the new async pattern along with your existing asynchronous implementation using the older model. Here is an example using the code bits used previously:

```
using (var connection = new SqlConnection("..."))
{
try
{
AsyncCallback callBack = new AsyncCallback(CallBack);
connection.OpenAsync().ContinueWith((task) =>
{
SqlCommand cmd = new SqlCommand("...", connection);
IAsyncResult ia = cmd.BeginExecuteNonQuery(callBack, cmd);
}, TaskContinuationOptions.OnlyOnRanToCompletion);
}
catch (SqlException)
{
//Log Exception
}
catch (Exception)
{
//Log Exception
}
}
```



Previously, asynchronous programming with the SqlClient data provider was achieved by adding Asynchronous Processing = true to the connection string and using SqlCommand.BeginExecuteNonQuery() or SqlCommand.BeginExecuteReader(). These methods still exist in .NET 4.5 (though you no longer need the asynchronous command in the connection string), but like elsewhere in the framework, the TPL has been implemented in SqlConnection, SqlCommand, SqlDataReader, and SqlBulkCopier (as well as their underlying abstract classes) to provide asynchronous versions of existing synchronous methods. The new methods follow the standard pattern of appending Async to the method name so, for example, SqlConnection.Open() now has as a corresponding asynchronous method SqlConnection.OpenAsync().
The following example demonstrates some of the async methods that are available.

```
class Program
{
public static void Main()
{
string connString = "Data Source=(local); Initial Catalog=DummyDb; Integrated Security=SSPI";
var cts = new CancellationTokenSource();
using (var conn = new SqlConnection(connString))
{
```




```
var command = new SqlCommand("WAITFOR DELAY '00:00:05';select FirstName from
FakePeople", conn);
ExecuteNonQueryAsync(conn, command)
.ContinueWith(t =>
{
Console.WriteLine("Start reader");
command.ExecuteReaderAsync(cts.Token)
.ContinueWith(async t2 =>
{
if (t2.Status == TaskStatus.Canceled)
Console.WriteLine("Read was cancelled");
while (await t2.Result.ReadAsync())
Console.WriteLine(t2.Result[0]);
});
Console.WriteLine("Do something else while getting the results");
});
Console.WriteLine("Waiting...");
Console.ReadKey();
}
}
public static async Task<int> ExecuteNonQueryAsync(SqlConnection conn, SqlCommand
cmd)
{
await conn.OpenAsync();
Console.WriteLine("Connection open");
await cmd.ExecuteNonQueryAsync();
Console.WriteLine("Query completed");
return 1;
}
}
```

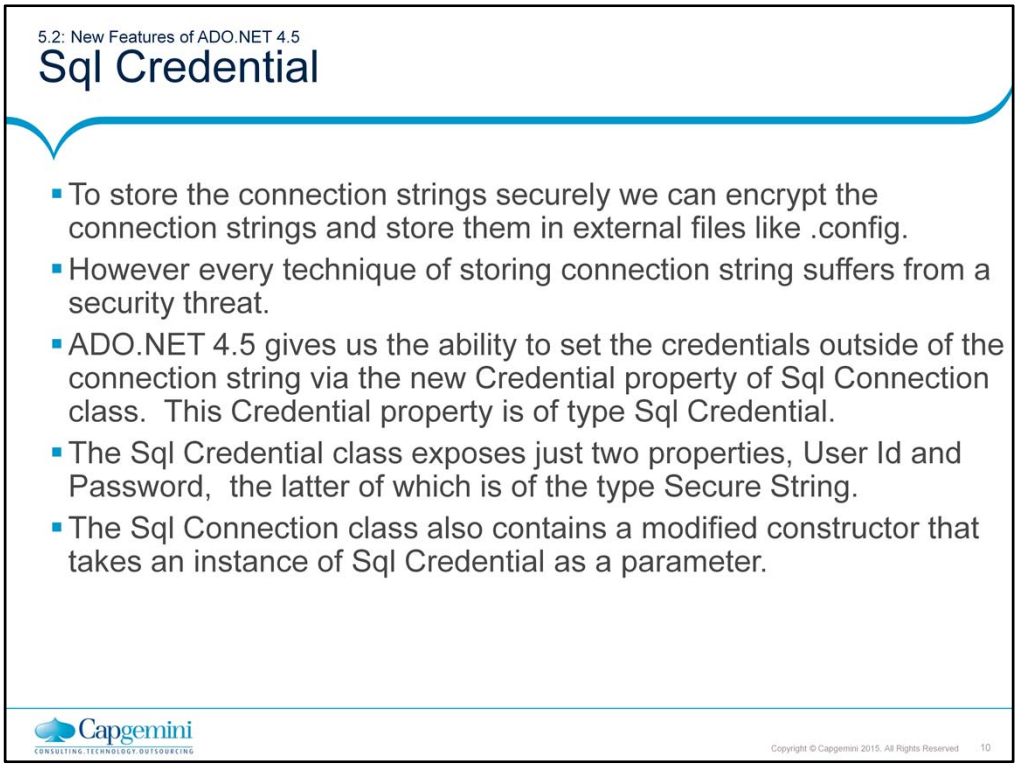
5.2: New Features of ADO.NET 4.5

Sql Credential

- Strings have always attracted interest of the hackers because
  - String data type is commonly used to store general purpose literals like connection string.
  - Large number of developers use string data type to store hard coded values like promotion codes, license keys etc.
- The information stored as string can be easily viewed by using tools like ILDASM or Windbg.
- Connection strings stored as strings are particularly vulnerable as they contain sensitive information like user name and password that will be used while connecting to database.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 9



Here is an example of how it can be used:

```
private void LoginButton_Click(object sender, RoutedEventArgs e)
{
var connString = "Data Source=(local);Initial Catalog=DummyDb";
var password = PasswordText.SecurePassword;
password.MakeReadOnly();
var sqlCredential = new SqlCredential(UsernameText.Text,password);
string message = "Successfully logged in";
try
{
using (var connection = new SqlConnection(connString,sqlCredential))
{
connection.Open();
}
}
catch (Exception ex)
{
message = "failed to log in";
}
MessageBox.Show(message);
}
```

The key thing to note is that the SqlCredential constructor will only allow you to pass an instance of SecureString that has been marked as read only.

Summary

- In this lesson, you have learnt:
  - New Features of ADO. NET 4.5, namely:
    - Asynchronous Data Access
    - SqlCredential

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 11

Review Question

- Question 1: The ___ and ___ modifiers in .NET Framework 4.5 are used for asynchronous data access.
- Question 2: Sql Credential class exposes two properties namely _____ and _____.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 12

ADO.NET 4.5

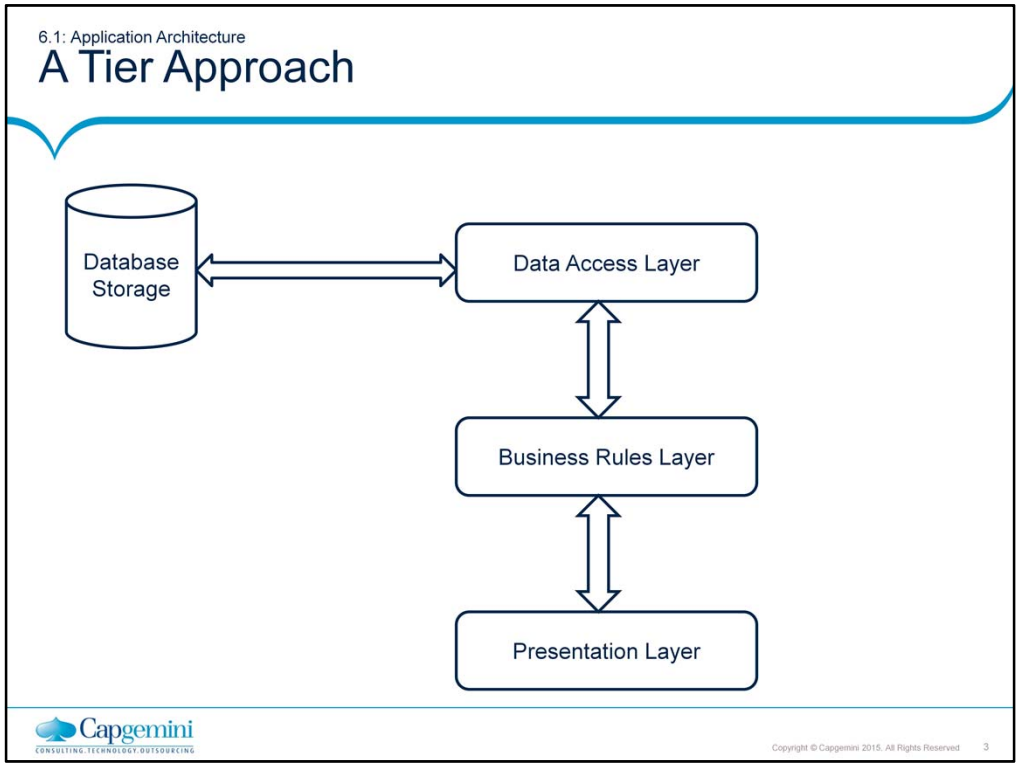
Lesson 6: Implementing Data Access Layer in ADO.NET

Lesson Objectives

- In this lesson, you will learn:
  - Application Architecture – Tier based architecture
  - Introduction to Data Access Layer
  - DAL – Design considerations
  - Data Access Layer Components
  - Writing code for implementing DAL Components

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 2

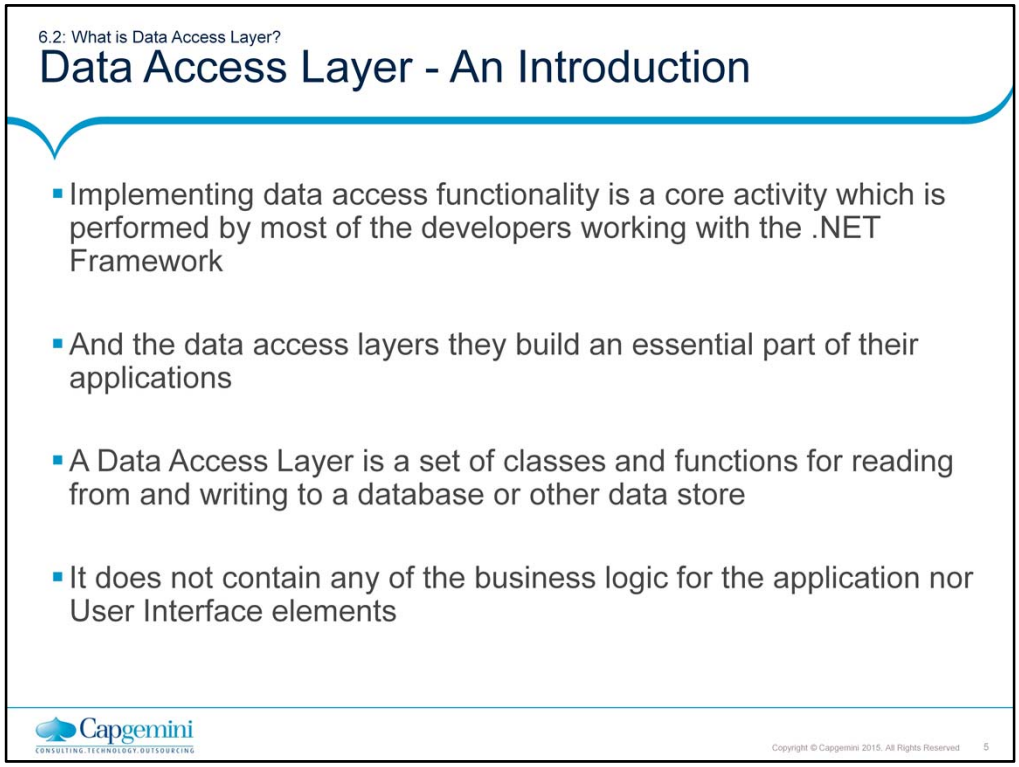


What is an Application Architecture?

Application architecture is the Organizational Design of an entire software application, including all sub-components and external applications interchanges. A software application is a system designed to automate specific tasks in a logical manner to satisfy a set of requirements. Software applications rely on underlying operating systems and databases to store and perform tasks within the application. The application architecture is the blueprint that defines how the software application will interact with servers and components within the domains of application layers.
There are three main Layers of control within all applications. These are the presentation layer, the business layer, and the data access layer. Each domain within an application has a specific responsibility that, when joined with the other layers, satisfy the underlying business requirements of an application.

Presentation Layer - This is the layer that provides an interface for the end user to your application. The presentation layer is a vitally important part of an application - an inappropriately architected presentation layer can lead to too much complexity, a lack of flexibility and an inefficient and frustrating user experience. It’s the layer which represents the interface between the user and the rest of the application.
Business Rules Layer - This is basically where the brains of your application reside; it contains things like the business rules, data manipulation, etc.

Data Access Layer - This layer is where you will write some generic methods to interface with your data. For example, we can write a method for creating and opening a Connection object (internal), and another for creating and using a Command object. This layer, obviously, contains no data business rules or data manipulation/transformation logic. It is merely a reusable interface to the database.

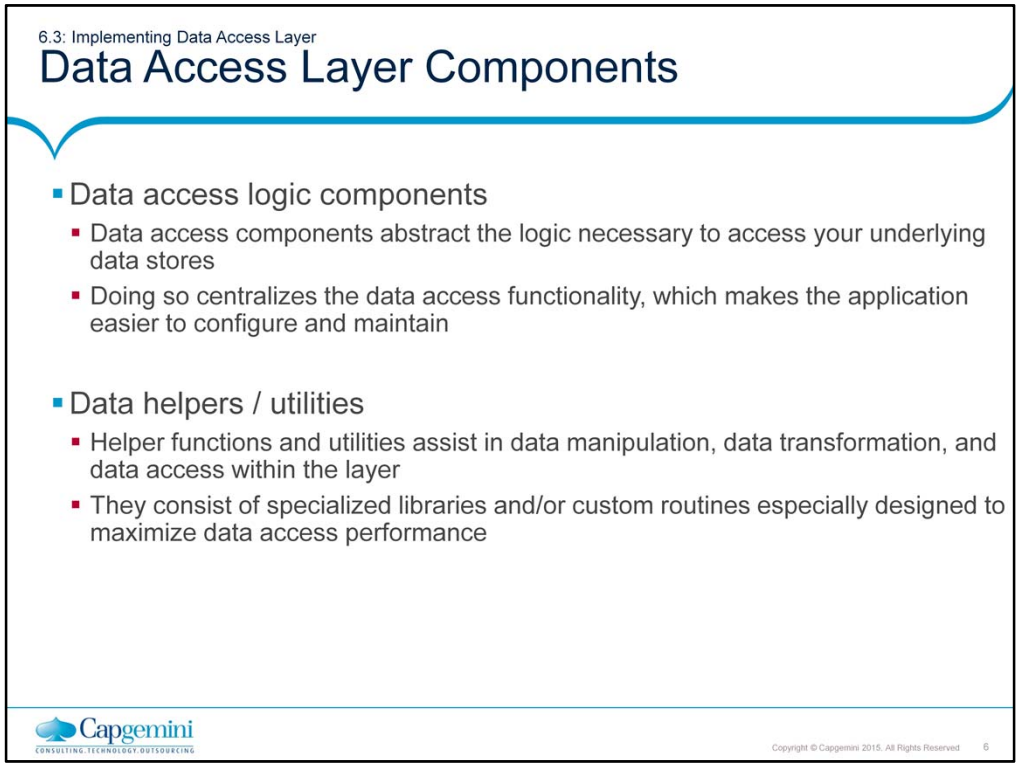


Data Access Layer (DAL) – An Introduction

Being an application developer, you have to design many UIs which comprises of Web Pages, Web Forms etc. that perform database CRUD (Create, Read, Update and Delete) operations directly from the code associated with the page. This is a quick but wrong and not so suggested approach that often results in applications that are difficult to maintain, and worse, tend to be inconsistent in their execution of error handling and transactional integrity.

By separating and centralizing code for the activities associated with the specific tasks like data access, you gain the ability to reuse the code, not only within a single project, but across multiple projects, as well. This can greatly simplify maintaining application logic since code for specific tasks is easy to find and changes only need to be made in one place.

In an ideal world, Web pages, or any app for that matter, should be unaware of how the data from a DAL is retrieved. They should only make polite requests to the DAL. It should not matter whether the data comes from Oracle, SQL Server.



Data Access Layer Components

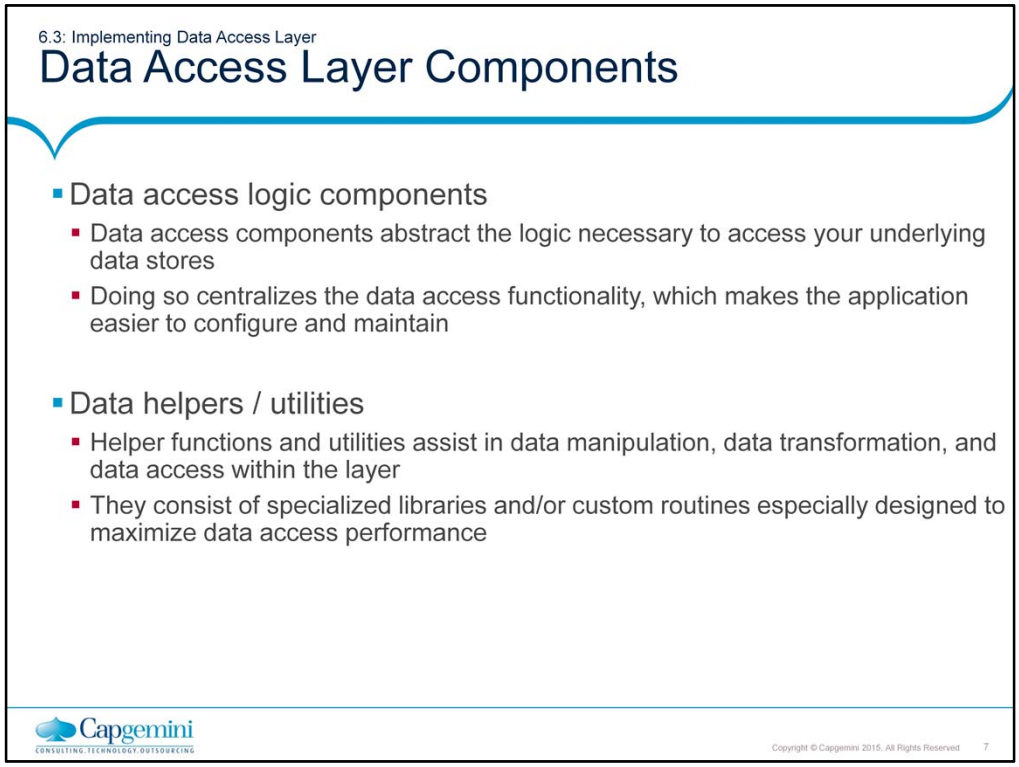
A correct approach to designing the data access layer will reduce development time and assist in maintenance of the data layer after the application is deployed. This part of the lesson briefly outlines an effective design approach for the data layer and the different important components of DAL.

A typical Data Access Layer consists of following main components.
Data access logic component

Data access components abstract the logic necessary to access your underlying data stores.
Doing so centralizes the data access functionality, which makes the application easier to configure and maintain.

Data helpers/utilities

Helper functions and utilities assist in data manipulation, data transformation, and data access within the layer.
They consist of specialized libraries and/or custom routines especially designed to maximize data access performance.



Data Access Layer Components

A correct approach to designing the data access layer will reduce development time and assist in maintenance of the data layer after the application is deployed. This part of the lesson briefly outlines an effective design approach for the data layer and the different important components of DAL.

A typical Data Access Layer consists of following main components.
Data access logic component

Data access components abstract the logic necessary to access your underlying data stores.
Doing so centralizes the data access functionality, which makes the application easier to configure and maintain.

Data helpers/utilities

Helper functions and utilities assist in data manipulation, data transformation, and data access within the layer.
They consist of specialized libraries and/or custom routines especially designed to maximize data access performance.

6.3: Implementing Data Access Layer

Data Access Layer Components

- Customer Management System:

Customer.cs:

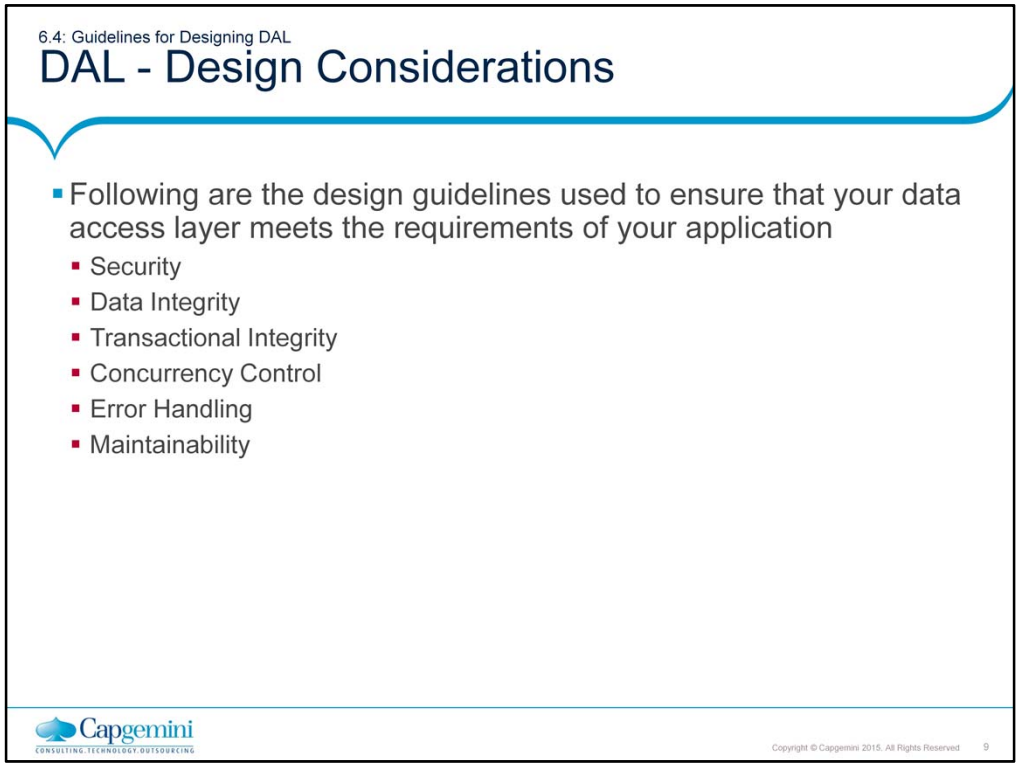
```
public class Customer
{
    //Properties to represent individual customer
   //E.g- CustomerID, CustomerName ect
}
```

CustomerOperations.cs:

```
public class CustomerOperations    //Data Access Logic Component
{
  public bool AddCustomerRecord(Customer customerObj)  //Data Helper/Utility
  {
      //Code to add the customer record.
  }
}
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 8



DAL - Design Considerations

The following design guidelines provide information about different aspects of the data access layer that you should consider. Follow these guidelines to ensure that your data access layer meets the requirements of your application, performs efficiently and securely, and is easy to maintain and extend as business requirements change.

Security – The security plays major role in designing DAL. We can design secure data access layer by creating separate database accounts for your applications with no direct permissions on tables within the database. This can be done by providing separate stored procedures for each Select, Update, Insert and Delete action that can be performed on an entity

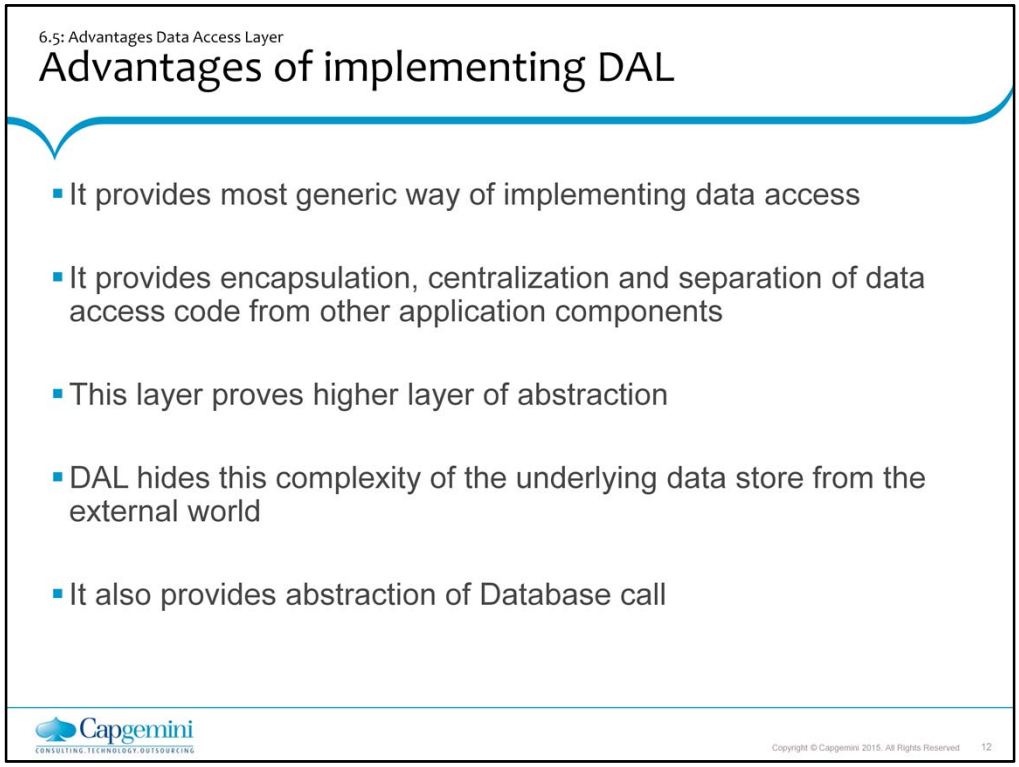
Data Integrity – We always have databases designed with all the necessary constraints to ensure the integrity or correctness of the data. However, we certainly don't want to wait until we submit the data to the database to find out that it doesn't meet the constraints, like referential integrity. Therefore, our DAL logic should be configured to reflect these same constraints and raise errors before we get to the database.

Transaction Integrity - If you are inserting multiple pieces of data in one or more tables, you may need to ensure that all the data is successfully saved. A DAL should support the ability to rollback a series of database transactions if any one of them fails.

Concurrency Control - When many people attempt to modify data in a database at the same time, a system of controls must be implemented so that modifications made by one person do not adversely affect those of another person. This is called concurrency control. We can handle concurrency in three different ways, first, we do nothing, allowing the changes of the last user to overwrite the changes of the first without warning. This is typically known as the "last in wins" scenario. Secondly, you can implement optimistic concurrency control when users do not lock data when they read it. When a user updates data, the system checks to see if another user changed the data after it was read. If another user updated the data, an error is raised. Typically, the user receiving the error rolls back the transaction and starts over. In the third option, you take a pessimistic stand. In this scenario, you predict that it is likely that two users will be vying for the same record so you place a lock on the record when the first person accesses it to prevent any others from accessing it until the first edit is complete.

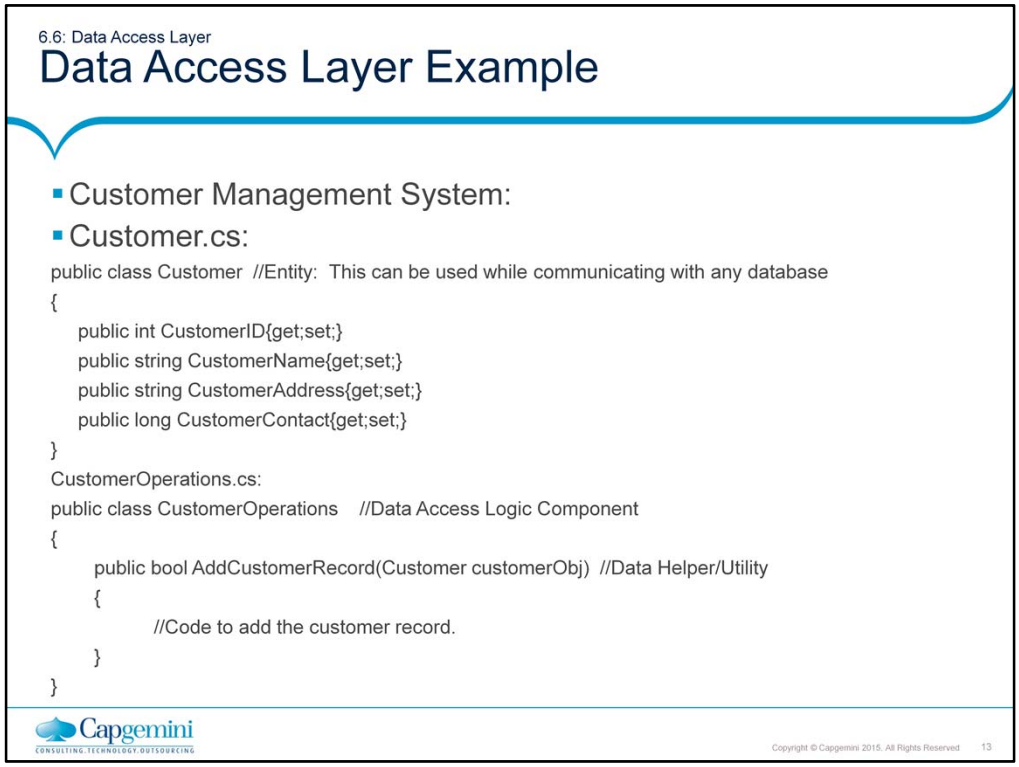
Error Handling - Since we want our DAL consumers to be unaware of the inner workings of our data retrieval, we certainly don't want unhandled SQL Server exceptions bubbling up to the presentation layer. Instead, the DAL should be providing more useful custom exceptions about the nature of the error. The DAL needs to present these exceptions to the consumer so that the consumer can decide whether it is OK to proceed, try again, or simply give up.

Maintainability – No application code is static or permanent. According to requirements we have to keep on changing application logic/code to meet challenges. Most of the times we needed to change or remove a field in a table and for that to happen we have to search through all our code to find every place that it is being referenced. Such types are changes are always required to be made in our application code. By centralizing your code, you limit the number of places this can happen. Also, by using data objects that provide hard-coded properties that match fields for our entities, the application will not compile if one is removed.



Advantages of implementing DAL

Applications using a data access layer can be either database server dependent or independent. If the data access layer supports multiple database types, the application becomes able to use whatever databases the DAL can talk to.
This layer encapsulates and compartmentalizes all our data access code within nice, clean components, that interact with our database.
The DAL might return a reference to an object complete with its attributes instead of a row of fields from a database table. With this benefit we can create client modules with a higher level of abstraction.
Instead of using commands such as insert, delete, and update to access a specific table in a database, a class and a few stored procedures could be created in the database. The procedures would be called from a method inside the class, which would return an object containing the requested values.
Business logic methods from an application can be mapped to the Data Access Layer. For example, instead of making a query into a database to fetch all users from several tables the application can call a single method from a DAL which abstracts those database calls.



In the above example, the developer is working on Customer Management System. The class called Customer has been designed which includes the attributes necessary to represent the customer. The object of this Customer class can be passed to a method that contains the necessary code to connect to the database and perform the CRUD operations. This method may contain the code to connect to SQL Server, Oracle, MS-Access or any other database. From this method, you can call a stored procedure residing on a database server to perform some operation. This relieves you from writing a query to perform a database operation. This method can be called from Business Logic Layer.

Summary

- In this lesson, you have learnt:
  - Understanding Application Architecture – N Tier Architecture
  - Understanding importance of implementing DAL in an application
  - DAL Design considerations
  - Understanding different components involved in DAL

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 14

Review Question

- Question 1 What are the different layers of Application?
- Question 2 Which are the two Data Access Layer Components?
- Question 3 From the following list, select the correct guidelines used while designing DAL
  - Data Integrity
  - Event Handling
  - Maintainability
  - Interoperability

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 15

Ado.Net

Lab Book

Document Revision History

Date	Revision No.	Author	Summary of Changes
22-June-2011	1	Ajit Jog	Content Creation
06-February-2015	2	Nachiket Inamdar	Addition of ADO.NET 4.5 features. Pending Approval.

Table of Contents

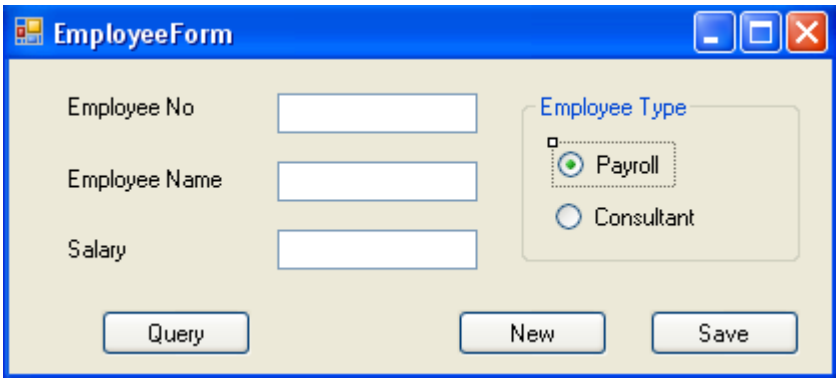
Lab 1. Using SqlCommand and SqlDataReader Classes

Description	In this Lab we will be retrieving employee information based on employee no and will be able to save new employee details
Goals	To Learn - • How to use sqlcommand object to execute a database stored procedure • How to use sqldatareader to read employee data • How to use sqlcommand to execute a DML statement
Time	180 Mins

The Table and SQL Server Stored Procedure used for this Lab:

```
/*
        create table employee
        ( empno int primary key,
          empname varchar(50) not null,
          empsal numeric(10,2) check(empsal >= 25000) ,
          emptype varchar(1) check(emptype in('C','P'))
        )
        go
        create proc GetEmployeeById
        (
                @eno int
        )
        as
                select * from employee where empno = @eno
        go
*/
```

1. Add a New Windows Form and design as below:



2. Define a connection object as form level member and write the following form load:


```
private void EmployeeForm_Load(object sender, EventArgs e)
 {
    con = new SqlConnection
                              (@"server=atrgsql\sql2005;database=labdemos;" +
                                                        "user id=sqluser;password=sqluser");
    con.Open();

  }
```

3. Add the following codes to the command buttons as below:

```
 private void btnquery_Click(object sender, EventArgs e)
 {
    try
    {
       SqlDataReader dreader=null;
       //The Procedure to execute
       SqlCommand cmd = new SqlCommand("GetEmployeeById", con);
       cmd.CommandType = CommandType.StoredProcedure;

       //define procedure parameter
       SqlParameter prm;
       prm = new SqlParameter();
       prm.SqlDbType = SqlDbType.Int;
       prm.Direction = ParameterDirection.Input;
       prm.ParameterName = "@eno";
       cmd.Parameters.Add(prm);

       //assign parameter value
       cmd.Parameters["@eno"].Value = int.Parse(txtempno.Text);

       //execute
       dreader = cmd.ExecuteReader();

       //if employee record found
       if (dreader.Read())
       {
          txtempname.Text = dreader["empname"].ToString();
          txtsalary.Text = dreader["empsal"].ToString();
          if (dreader["emptype"].ToString() == "P")
             rdpayroll.Checked = true;
          else
             rdconsultant.Checked = true;
       }
       else
       {
          btnnew_Click(btnnew, e);
          MessageBox.Show("No such employee");
       }
       dreader.Close();
    }
```



```
    catch (SqlException sqlex)
    {
        MessageBox.Show(sqlex.Message);
    }
}

private void btnsave_Click(object sender, EventArgs e)
{
    try
    {
        //The Insert DML to add employee record
        SqlCommand cmd = new SqlCommand
          ("insert into employee values(@eno,@enm,@esal,@etyp)", con);

        //The Parameters
        cmd.Parameters.Add("@eno", SqlDbType.Int);
        cmd.Parameters.Add("@enm", SqlDbType.VarChar, 50);
        cmd.Parameters.Add("@esal", SqlDbType.Decimal);
        cmd.Parameters.Add("@etyp", SqlDbType.VarChar, 1);

        //Assigning Values to parameters
        cmd.Parameters["@eno"].Value = txtempno.Text;
        cmd.Parameters["@enm"].Value = txtempname.Text;
        cmd.Parameters["@esal"].Value = txtsalary.Text;
        cmd.Parameters["@etyp"].Value = rdpayroll.Checked == true ? "P" : "C";

        //Execute Insert ....
        cmd.ExecuteNonQuery();
        MessageBox.Show("Employee Details Saved");
    }
    catch (SqlException sqlex)
    {
        MessageBox.Show(sqlex.Message);
    }
}

private void btnnew_Click(object sender, EventArgs e)
{
    txtempno.Text = "";
    txtempname.Text = "";
    txtsalary.Text = "";
    txtempno.Focus();
}
```

4. Run the Application
    a. Click New to clear the form if textboxes are already populated
    b. Fill the Employee Form and click Save to add new employee record.
    c. Click New to clear the form and type in an Employee No and click Query.

Assignment 1

To Do:
Add the delete button on the form. If the user clicks delete button ask for confirmation and if user confirms then delete the employee record.

Assignment 2

To Do:
Create a SQL Server stored procedure to add a new employee record. The procedure should accept all the employee details as parameter except empno. Procedure should auto generate next sequential empno and return that as well to the caller. [Hint: Use Output parameter]. Rewrite the btnsave click code so as to call this stored procedure while adding the new employee record.

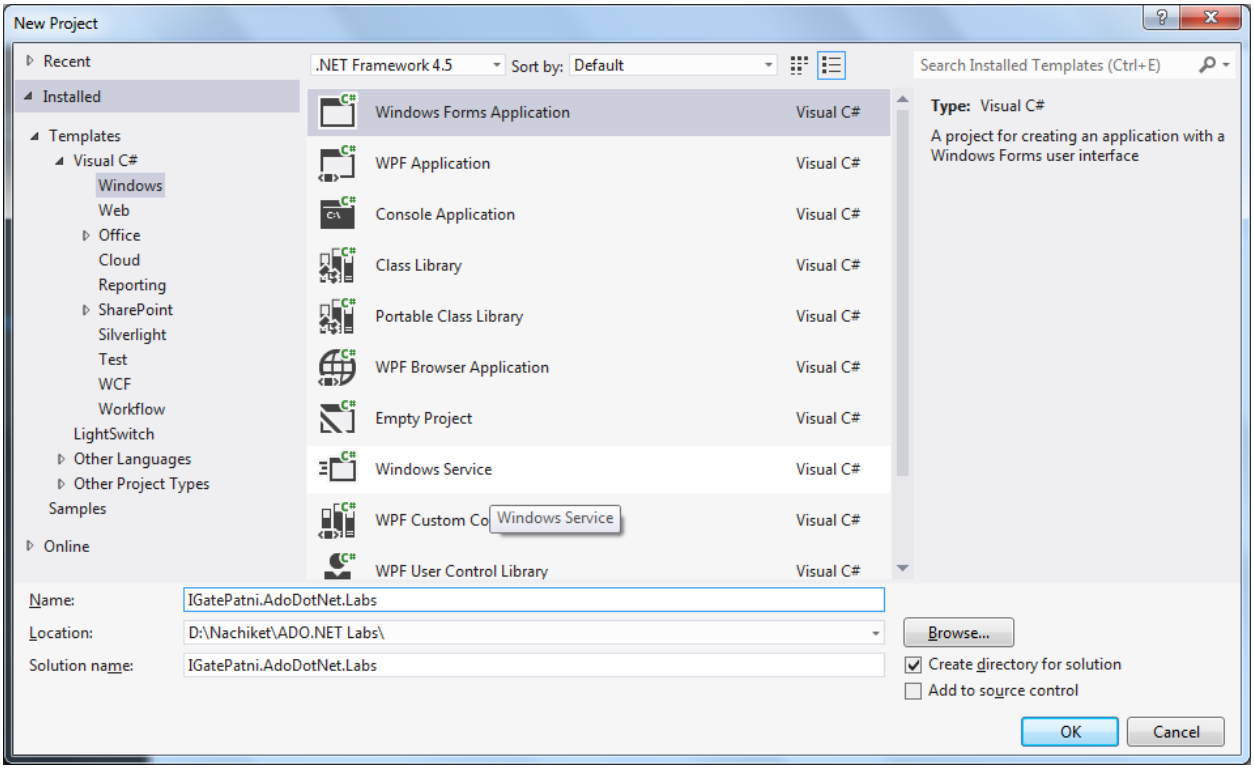
Lab 2. DataSet and DataAdapter

Description	We will be using DataAdapter and DataSet classes to retrieve information about application users from database. User can scroll as well edit the details and save the changes back.
Goals	To Learn - • How to use DataAdapter to retrieve relational data • Use DataSet to store and display data • Save the changes back to database
Time	60 Mins

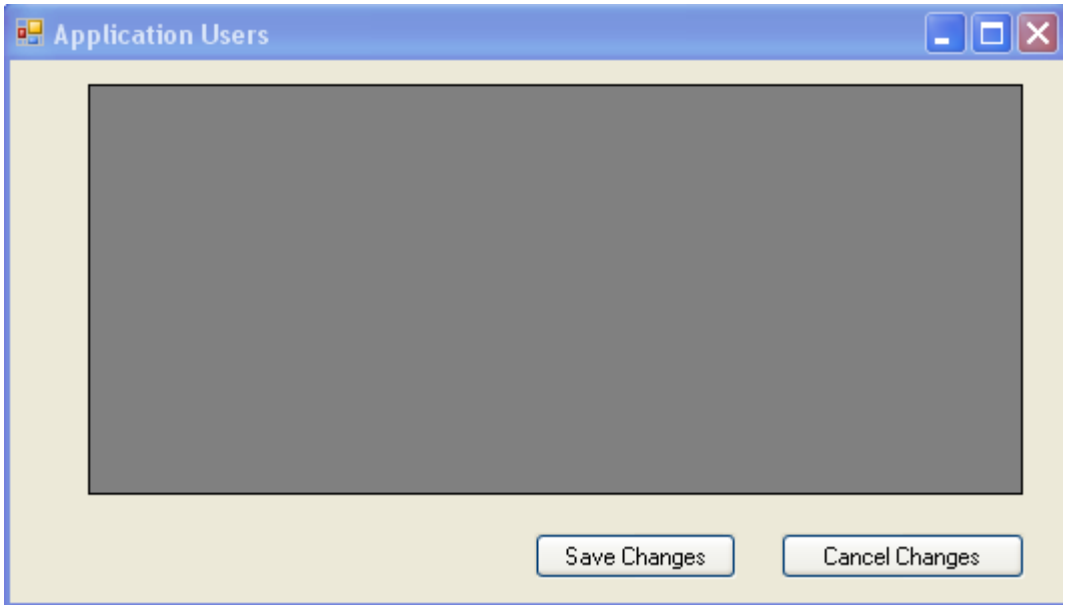
The Database Table used in this Lab:

```
/*
        create table applicationusers
        (
                userid varchar(10) primary key,
                username varchar(30) not null,
                city varchar(30) not null,
                password varchar(30) check(len(password) >5)
        )
*/
```

1. Create a new Windows Application Project , Name it IGatePatni.AdoDotNet.Labs



2. Design the Form as below:
    a. Drag GridView (Name: grdUsers) and 2 Buttons
    b. Rename the Form1 to DataSetAdapterDemo



3. Include the following namespace in code behind of the form:

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

4. Add the following as Form level members (inside form class below constructor definition)

```
SqlConnection con;
SqlDataAdapter da;
DataSet ds;
```

5. The Form Load code:

```
private void DataSetAdapterDemo_Load(object sender, EventArgs e)
 {
    con = new SqlConnection
       (@"server=atrgsql\sql2005;database=labdemos;" +
                          "user id=sqluser;password=sqluser");

    con.Open();
    ds = new DataSet();
    //select - For Data Retrieval
    da = new SqlDataAdapter("select * from applicationusers", con);

    //So that we should be able to save changes back to database....
    SqlCommandBuilder bld = new SqlCommandBuilder(da);

    da.Fill(ds, "appusers");

    grdUsers.DataSource = ds.Tables["appusers"];

 }
```

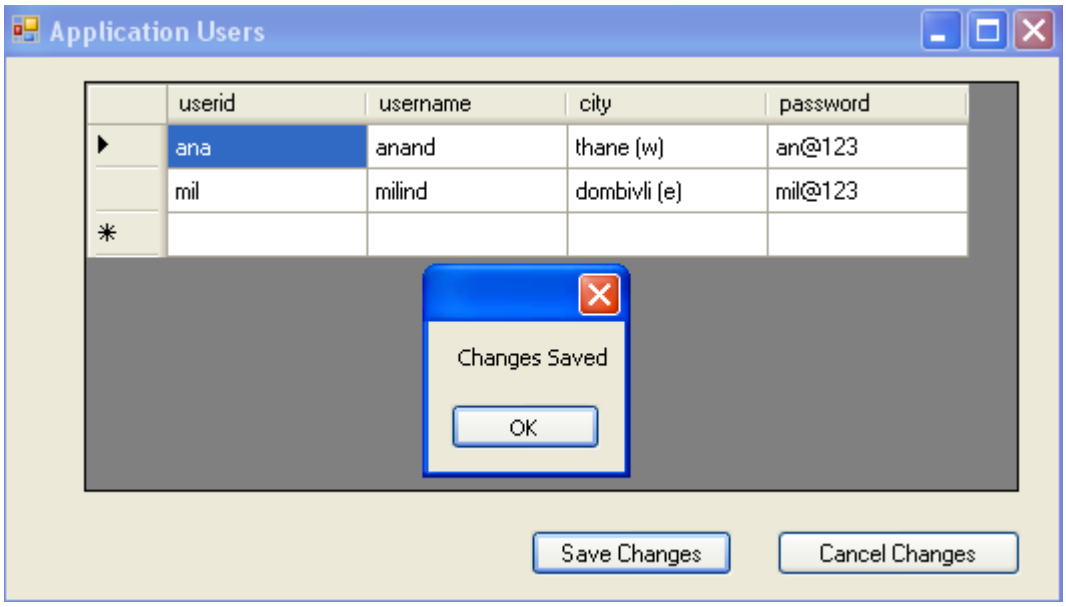
6. The Cancel and Save Button Code:

```
 private void btncancel_Click(object sender, EventArgs e)
 {
    ds.Tables["appusers"].RejectChanges();
  }

 private void btnsave_Click(object sender, EventArgs e)
 {
    try
    {
       //Save Changes to Database
       da.Update(ds.Tables["appusers"]);
       MessageBox.Show("Changes Saved");
    }
    catch (SqlException sqlex)
    {
       MessageBox.Show(sqlex.Message);
    }
```



```
}
```

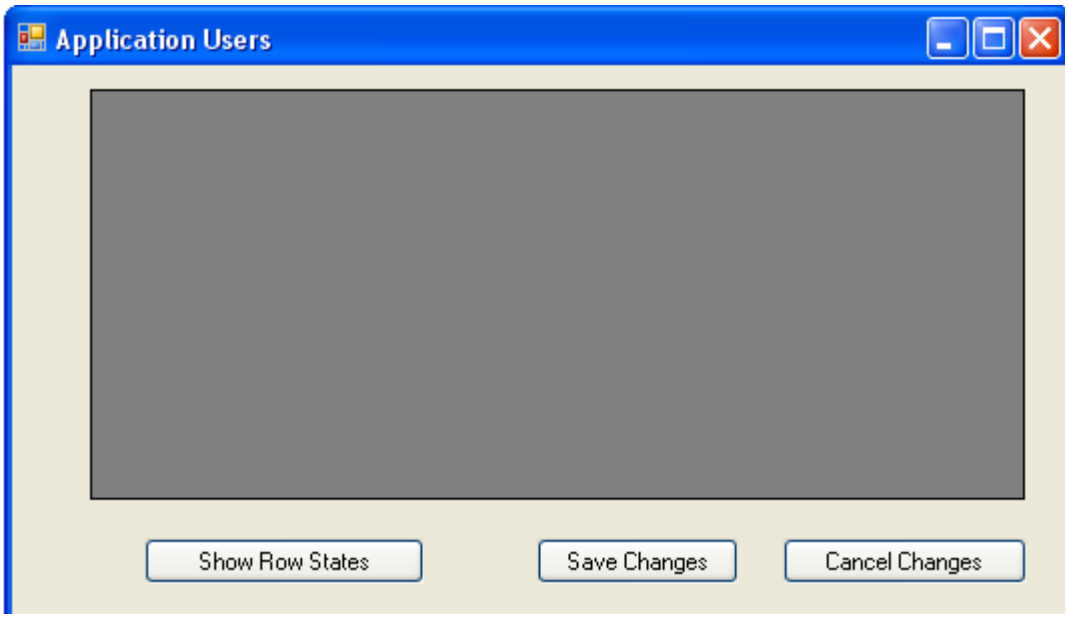
7. Run the application
    a. makes changes in the Grid and click cancel
    b. again make modification and click save button
8. Close and rerun the application to ensure that the changes are saved.



Lab 3. Understand RowState Concept for Data Rows of DataTable

Description	In the previous Lab we will be adding extra code to iterate Data Rows and check their row states.
Goals	To Learn - • Understand RowState Concept • See how the row states changes according to the changes made by the end user; to track it.
Time	30 Mins

1. In the previous lab drag one more button (btnstate) “Show Row States” on the form as below:



2. Write the following code:

```
private void btnstates_Click(object sender, EventArgs e)
    {
        //Iterate through Rows of DataTable and display RowStates .....

        foreach (DataRow drow in ds.Tables["appusers"].Rows)
        {
            if (drow.RowState == DataRowState.Deleted)
            {
                MessageBox.Show(
                    drow["username",DataRowVersion.Original].ToString() + " deleted ");
```



```
            }
            else
               MessageBox.Show(drow["username"].ToString()+ "  "
                        + drow.RowState.ToString() );
        }
    }
```

3. Run the Program check it following test cases:
    a. make changes to the first row and click the above button
    b. click cancel button and recheck the rowstates
    c. make changes to 2nd row and delete 1st row and check their rows states, and then click cancel changes button the deleted row should come back.
    d. Now make changes to both rows and click save changes and check the row states.

Lab 4. Using DataView Object to Filter DataTable

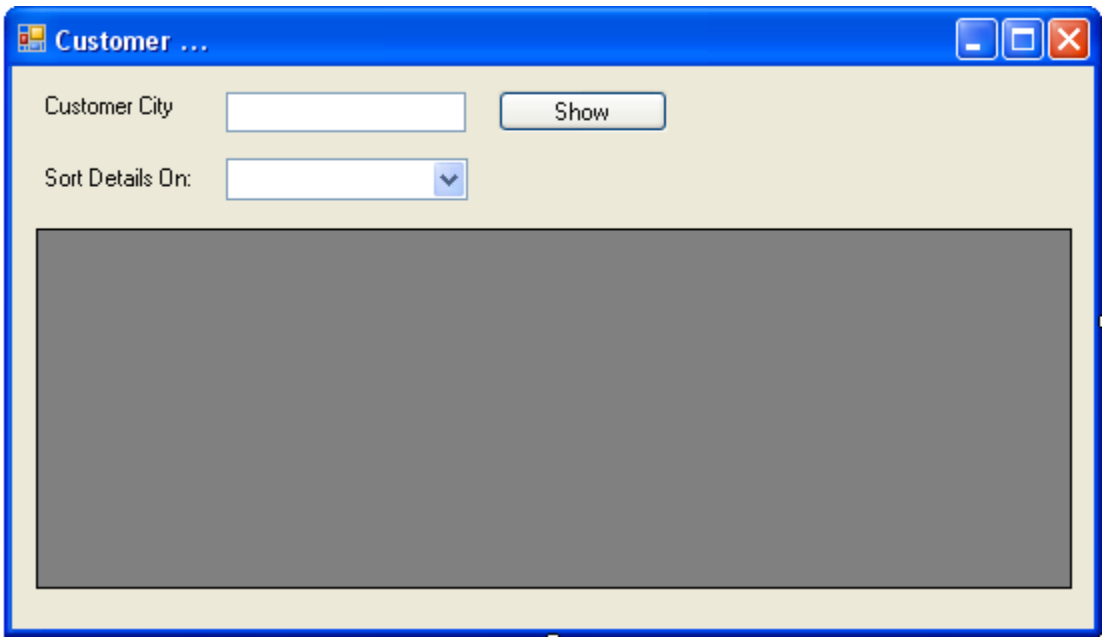
Description	In this Lab we will be displaying customer details and allow the user to filter the customer information on city.
Goals	To Learn - • Understand how to use DataView to do client side filtering and sorting.
Time	90 Mins

The Database Table Structure used in this Lab:

```
/*
        create table customer
        (
                customerid      int identity primary key,
                customername varchar(50),
                city    varchar(30),
                creditlimit     numeric(10,2)
        )
 */
```

1. Add a new blank windows form in the project
   a. Project => Add Windows Form, Name it CustomerForm

2. Drag the controls and design the form as below:
   a. TextBox (txtcity), Button (btnshow), ComboBox (cmbcolumnlist) and grid grdCustomers

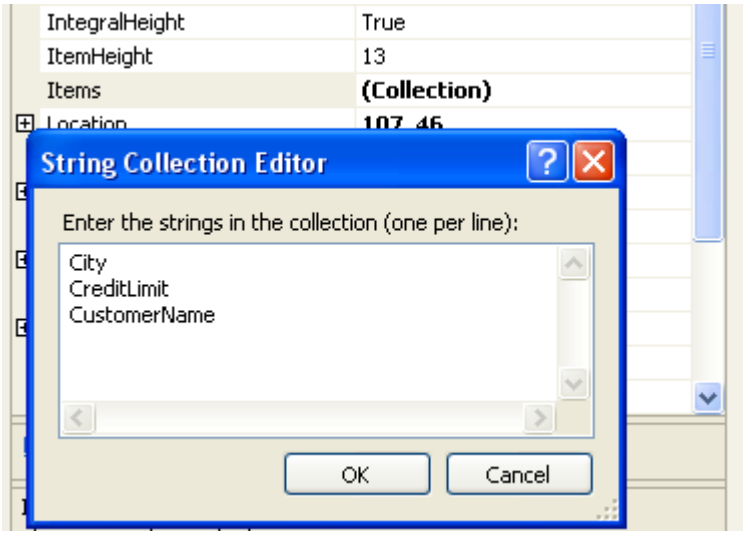


3. In the code behind include using **System.Data.SqlClient;** at the top
4. Define the following Ado.net objects at the form level below:

   SqlConnection **con;**
   SqlDataAdapter **da;**
   DataSet **ds;**
5. Form Load event code is given

```
private void CustomerForm_Load(object sender, EventArgs e)
{
   con = new SqlConnection
      (@"server=atrgsql\sql2005;database=labdemos;" +
                        "user id=sqluser;password=sqluser");

   con.Open();
   ds = new DataSet();
   //select - For Data Retrieval
   da = new SqlDataAdapter("select * from customer", con);

   da.Fill(ds, "cust");

   grdCustomers.DataSource = ds.Tables["cust"];
}
```

6. Go to Items property of ComboBox and add the following options



7. Set DropDownStyle property of ComboBox to “DropDownList”
8. The ComboBox “SelectedIndexChanged” event code

```
private void cmbcolumnlist_SelectedIndexChanged(object sender, EventArgs e)
{
        ds.Tables["cust"].DefaultView.Sort = cmbcolumnlist.Text;
 }
```

9. The Show Button Code:

```
private void btnshow_Click(object sender, EventArgs e)
{
    ds.Tables["cust"].DefaultView.RowFilter ="City like '" + txtcity.Text + "'";
}
```

10. Make this form as startup in Program.cs
11. Run the Program
    a. Type “dom*” in city textbox and click show
    b. Type “*n*” in city textbox and click show
    c. Select different column names from combo and see the sorted data in grid.

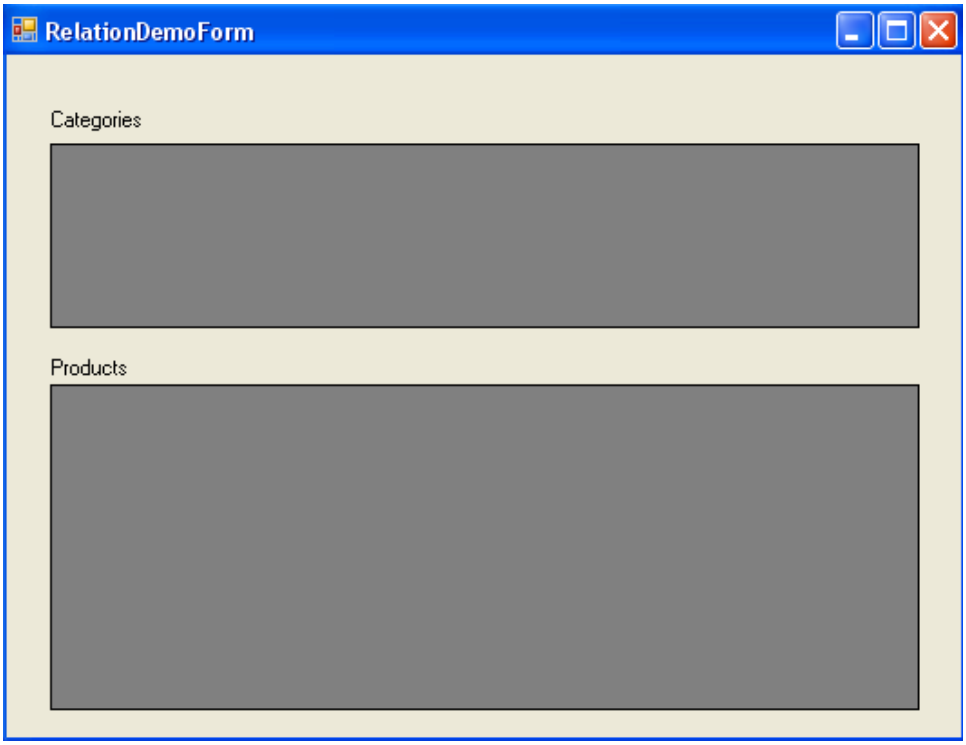
Assignment 3

To Do:
Create a table Supplier with coulmns SupplierId (primary key), Suppliername, City, ContactNo, CreditBalance. Create a Windows Form which will display the all the suppier details in Grid. Allow user to filter the suppliers based on City or Name. The user should also be able to make changes to the supplier details and save the changes back to database.

Lab 5. Using DataRelation Class

Description	In this Lab we will be filling a dataset with categories and products information from database and making a parent-child relationship within dataset.
Goals	To Learn - • Understand how to use establish a master-detail relationship between 2 Data Tables in a DataSet
Time	60 Mins

1. Add a New Windows Form and design as below:
    a. DataGridViews: grdCategories, grdProducts



2. Add the following in the Form Class Code
3. This Code
    a. Fills the DataSet with Categories, Products Details.
    b. Creates a DataRelation based on common column (Categoryid in this case) and adds it to Relations Collection of Dataset.
    c. Then sets the DataSource of both the grids. (Note the way the datasource is set for grid “grdProducts”)
    d. Sets the Update and Delete Cascade Rules.

```
//FORM LEVEL MEMBERS
```



```
SqlConnection con = new SqlConnection
                    (@"server=atrgsql\sql2005;database=labdemos;" +
                "user id=sqluser;password=sqluser");
SqlDataAdapter dacat, daprod;
DataSet ds_cat_pro;

private void RelationDemoForm_Load(object sender, EventArgs e)
{
   dacat = new SqlDataAdapter("select * from category", con);
   daprod = new SqlDataAdapter("select * from product", con);
   con.Open();
   ds_cat_pro   = new DataSet();
   dacat.Fill(ds_cat_pro, "cat");
   daprod.Fill(ds_cat_pro, "pro");
   con.Close();
   //Setting Default Constraint

   ds_cat_pro.Tables["pro"].Columns["categoryid"].DefaultValue = 1;

   //CREATING RELATION BETWEEN THE TWO DATA TABLES
   DataRelation dre1 = new DataRelation("catpro_relation",

    ds_cat_pro.Tables["cat"].Columns["CategoryId"],

    ds_cat_pro.Tables["pro"].Columns["CategoryId"]);
   ds_cat_pro.Relations.Add(dre1);

   ds_cat_pro.Relations["catpro_relation"].ChildKeyConstraint.DeleteRule
                                                          = Rule.None;
   ds_cat_pro.Relations["catpro_relation"].ChildKeyConstraint.UpdateRule
                                                          = Rule.None;

   //DIFFERENT CONSTRAINT RULE OPTIONS:

   //NONE              WILL NOT ALLOW DELETING MASTER RECORD
   //CASCADE           WILL DELETE MASTER AS WELL AS CHILD RECORDS
   //SETDEFAULT:     WILL ALLOW DELETION OF MASTER RECORD AND SET THE
VALUE  IN CHILD WHICH IS DEFAULT
   //SETNULL:          WILL SET VALUE OF COLUMN TO NULL IN CHILD TABLE AND
DELETE RECORD FROM MASTER TABLE

   grdCategories.DataSource = ds_cat_pro.Tables["cat"];

   grdProducts.DataSource = ds_cat_pro.Tables["cat"];
   grdProducts.DataMember = "catpro_relation";
}
```

4. Run the Application
    a. The Grid will be loaded with category and product details
    b. If you navigate category records the corresponding product details will be automatically displayed.

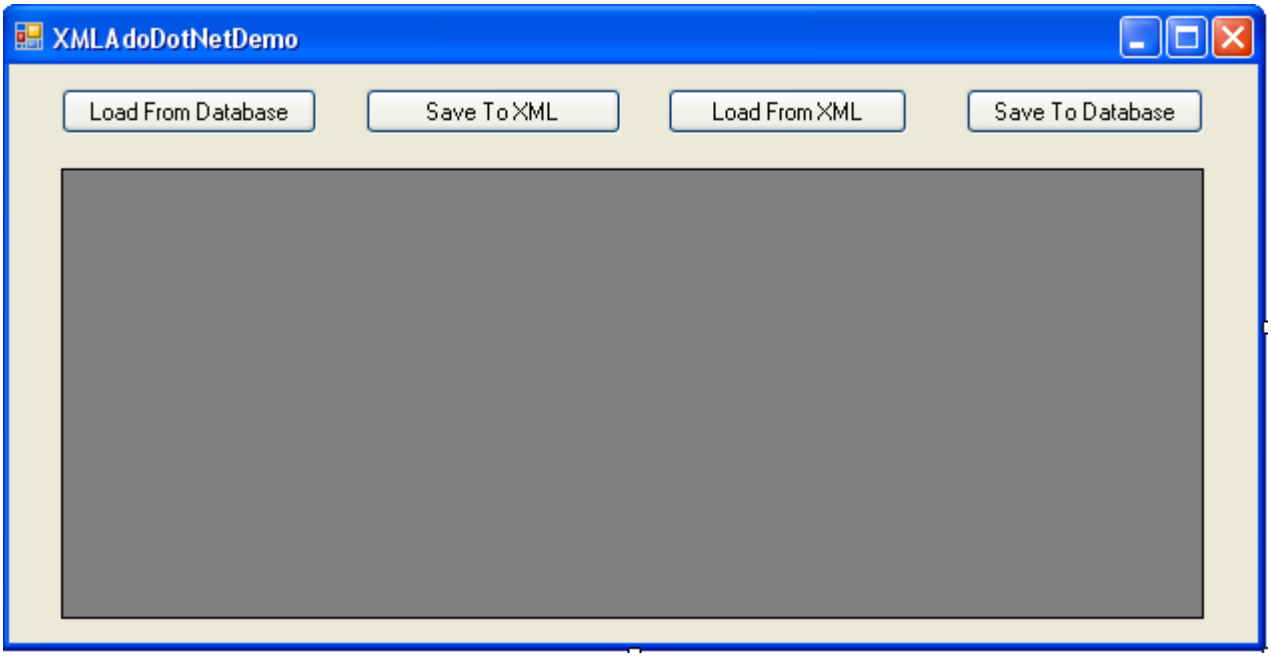
c. Try deleting a category by selecting the whole Grid Row and pressing delete key, it will fail because of delete rule set to none.
d. Close the application

5. Change the delete rule to cascade and repeat the above step.

Lab 6. Using XML support in Ado.Net

Description	In this lab we will be retrieving and storing product details in a xml file and save the schema structure in a separate xsd file.
Goals	To Learn - • How to use the xml support provided in ado.net to cache the product details in local xml files and use it. • Performing batch updates to database while developing disconnected applications • How xml support can be used for disconnected scenario.
Time	60 Mins

1. Design a Windows Form as below: Name the form XMLAdoDotNetDemo



2. Add these namespaces

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

3. Define these instances as form members

```
SqlConnection con;
SqlDataAdapter daprod;
DataSet dsprod;
```

4. The form load code:


```
private void XMLAdoDotNetDemo_Load(object sender, EventArgs e)
{
  string constr = ConfigurationManager.ConnectionStrings
                                        ["labdemoconnectstring"].ConnectionString;
        con = new SqlConnection(constr);
  con.Open();
        daprod = new SqlDataAdapter("select * from product", con);
  SqlCommandBuilder cb = new SqlCommandBuilder(daprod);
        dsprod = new DataSet();
}
```

5. The “Load From Database” Button Code:

```
private void btnloadfromdb_Click(object sender, EventArgs e)
{
   daprod.Fill(dsprod, "prod");
   grdproducts.DataSource = dsprod.Tables["prod"];
   btnloadfromdb.Enabled = false;
 }
```

6. The “Save To XML” Button Code:

```
   private void btnsavetoxml_Click(object sender, EventArgs e)c
{
    //Save Schema and Data into xml file
   dsprod.WriteXmlSchema(@"D:\products.xsd");
   dsprod.WriteXml(@"D:\products.xml", XmlWriteMode.DiffGram);
   MessageBox.Show("Data saved to disk");
 }
```

7. The “Save To Database” Button Code:

```
private void btnloadfromxml_Click(object sender, EventArgs e)
 {
        //Read Data from xml file into Data Set
 dsprod.ReadXmlSchema(@"D:\products.xsd");
        dsprod.ReadXml(@"D:\products.xml",XmlReadMode.DiffGram);
 grdproducts.DataSource = dsprod.Tables["prod"];
 }
```

8. The “Load From XML” Button Code:

```
private void btnsavetodb_Click(object sender, EventArgs e)
{
    try
        {
            daprod.Update(dsprod.Tables["prod"]);
                MessageBox.Show("Changes Saved");
        }
   catch (SqlException sqlex)
```



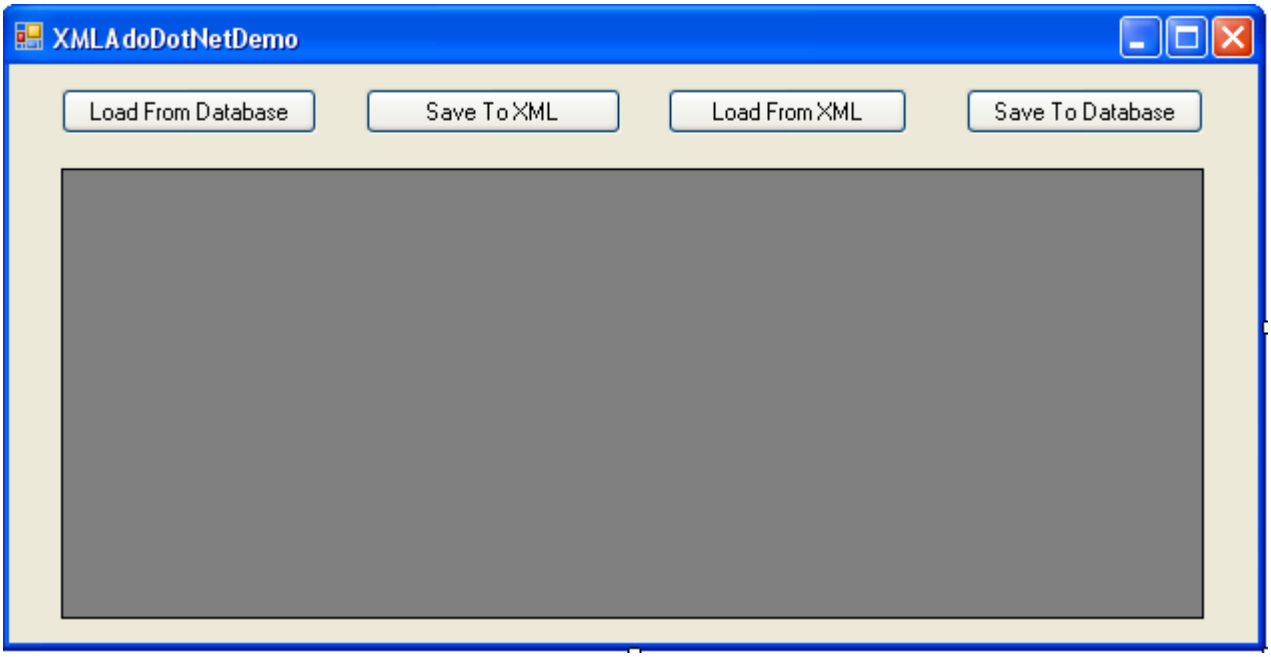
```
            {
                MessageBox.Show(sqlex.Message);
            }
}
```

9. Run the application and test:
    a. click the “load from database” button the grid will be populated with product details
    b. make changes to few rows and click “save to xml” button (saving data to local cache)
    c. close the application and re-run it.
    d. click on the “load from xml” to load the product details from xml (ie: locally cached information)
    e. click “save to database” to save changes back to database.

Lab 7. Using XML support in Ado.Net

Description	In this lab we will be retrieving and storing product details in a xml file and save the schema structure in a separate xsd file.
Goals	To Learn - • How to use the xml support provided in ado.net to cache the product details in local xml files and use it. • Performing batch updates to database while developing disconnected applications • How xml support can be used for disconnected scenario.
Time	60 Mins

1. Design a Windows Form as below: Name the form XMLAdoDotNetDemo



2. Add these namespaces

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

3. Define these instances as form members

```
SqlConnection con;
SqlDataAdapter daprod;
DataSet dsprod;
```

4. The form load code:


```
private void XMLAdoDotNetDemo_Load(object sender, EventArgs e)
{
  string constr = ConfigurationManager.ConnectionStrings
                                        ["labdemoconnectstring"].ConnectionString;
        con = new SqlConnection(constr);
  con.Open();
        daprod = new SqlDataAdapter("select * from product", con);
  SqlCommandBuilder cb = new SqlCommandBuilder(daprod);
        dsprod = new DataSet();
}
```

5. The “Load From Database” Button Code:

```
private void btnloadfromdb_Click(object sender, EventArgs e)
{
   daprod.Fill(dsprod, "prod");
   grdproducts.DataSource = dsprod.Tables["prod"];
   btnloadfromdb.Enabled = false;
 }
```

6. The “Save To XML” Button Code:

```
   private void btnsavetoxml_Click(object sender, EventArgs e)c
{
    //Save Schema and Data into xml file
   dsprod.WriteXmlSchema(@"D:\products.xsd");
   dsprod.WriteXml(@"D:\products.xml", XmlWriteMode.DiffGram);
   MessageBox.Show("Data saved to disk");
 }
```

7. The “Save To Database” Button Code:

```
private void btnloadfromxml_Click(object sender, EventArgs e)
 {
        //Read Data from xml file into Data Set
 dsprod.ReadXmlSchema(@"D:\products.xsd");
        dsprod.ReadXml(@"D:\products.xml",XmlReadMode.DiffGram);
 grdproducts.DataSource = dsprod.Tables["prod"];
 }
```

8. The “Load From XML” Button Code:

```
private void btnsavetodb_Click(object sender, EventArgs e)
{
    try
        {
            daprod.Update(dsprod.Tables["prod"]);
                MessageBox.Show("Changes Saved");
        }
  catch (SqlException sqlex)
```



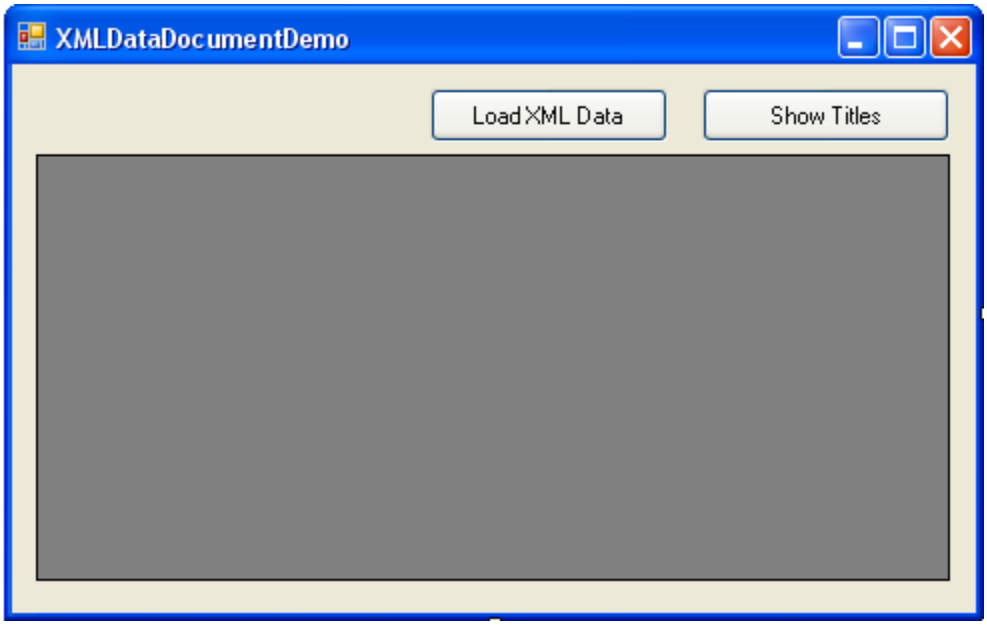
```
        {
            MessageBox.Show(sqlex.Message);
        }
}
```

9. Run the application and test:
    a. click the “load from database” button the grid will be populated with product details
    b. make changes to few rows and click “save to xml” button (saving data to local cache)
    c. close the application and re-run it.
    d. click on the “load from xml” to load the product details from xml (ie: locally cached information)
    e. click “save to database” to save changes back to database.

Lab 8. Using XmlDataDocument object

Description	In this lab we will be using XmlDataDocument object to load xml data and expose and work with it in Relational representation in form of DataSet and hierarchical representation as xml dom.
Goals	To Learn - • How to use the XmlDataDocument object so that we can work with data in relational as well as hierarchical representation model.
Time	60 Mins

1. Add a new windows form and design as below:
2. Drag 2 buttons (btnload, btnshowtitles) and a grid (grdbook)



3. Set enabled property of “show titles” button to false.
4. Add a XSD file
    a. Project => Add New Item ; Select “XML Schema” Template
    b. Name it: BookStore.xsd
5. Put the following schema definition in the file:

```
<?xml version="1.0" encoding="utf-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <xsd:element name="bookstore" type="bookstoreType"/>

  <xsd:complexType name="bookstoreType">
    <xsd:sequence maxOccurs="unbounded">
```

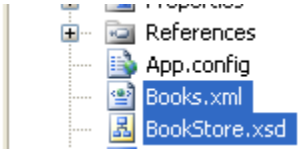


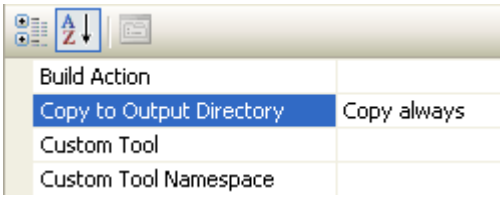
```
    <xsd:element name="book"  type="bookType"/>
   </xsd:sequence>
  </xsd:complexType>

  <xsd:complexType name="bookType">
   <xsd:sequence>
    <xsd:element name="title" type="xsd:string"/>
    <xsd:element name="author" type="authorName"/>
    <xsd:element name="price"  type="xsd:decimal"/>
   </xsd:sequence>
   <xsd:attribute name="genre" type="xsd:string"/>
  </xsd:complexType>

  <xsd:complexType name="authorName">
   <xsd:sequence>
    <xsd:element name="first-name"  type="xsd:string"/>
    <xsd:element name="last-name" type="xsd:string"/>
   </xsd:sequence>
  </xsd:complexType>

</xsd:schema>
```

6. Add a XML Data file
   a. Project => Add New Item ; Select “XML File” Template
   b. Name it: Books.xml
7. Following is xml data in it:

```
<?xml version="1.0" encoding="utf-8" ?>

<bookstore>
  <book genre='novel' ISBN='10-861003-324'>
   <title>The Handmaid's Tale</title>
   <price>19.95</price>
  </book>
  <book genre='fiction' ISBN='10-999003-122'>
   <title>The Voyager's Tale</title>
   <price>29.95</price>
  </book>
  <book genre='fiction' ISBN='21-595002-152'>
   <title>The Robotic Man</title>
   <price>25.95</price>
  </book>
  <book genre='novel' ISBN='1-861001-57-5'>
   <title>Pride And Prejudice</title>
   <price>24.95</price>
  </book>
</bookstore>
```

8. Select both BookStore.xsd and Books.xml at once in solution explorer.
9. Go to properties window (Press F4)
   a. Set property “Copy to Output Directory” value to: copy always





10. Go to code behind add the following namespaces
    using **System.Data;**
    using **System.Xml;**
11. The Button event code:

```
    private void btnload_Click(object sender, EventArgs e)
{
        doc = new XmlDataDocument();

      // Load the schema file.
        doc.DataSet.ReadXmlSchema("BookStore.xsd");

  // Load the XML data.
  XmlTextReader reader = new XmlTextReader("Books.xml");
  // Move the reader to the root node and load xml data
  reader.MoveToContent();
  doc.Load(reader);

  // Update the price on the first book using the DataSet methods.
        // Working with data as relational model
  DataTable books = doc.DataSet.Tables["book"];
  books.Rows[0]["price"] = "101.95";

  grdbook.DataSource = doc.DataSet.Tables["book"];
      btnload.Enabled = false;
  btnshowtitles.Enabled = true;
 }

 private void btnshowtitles_Click(object sender, EventArgs e)
{
    // Get the node list of titles of type 'novel'
   // Working with data as heirarchical model
     XmlNodeList nodelist=  doc.DocumentElement.SelectNodes
                                          ("//title[../@genre = 'novel']");
     string msg = "";
     foreach (XmlNode node in nodelist)
  {
          msg += node.InnerText + "\n";
  }
  MessageBox.Show(msg);
 }
```

12. Make this form as startup and run.

a. Click the Load XML Data Button and then click show titles button

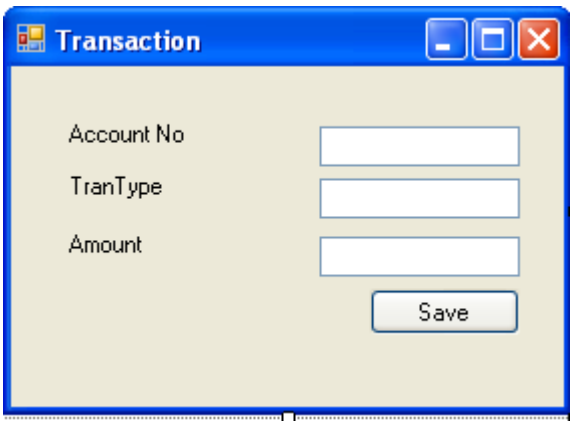
Lab 9. Using Transaction for ATOMIC database operation

Description	In this lab we will be using Account Master (AccMaster Table) and Transaction Table (AccTran Table). The user will be able to deposit and withdraw amount from the available accounts.
Goals	To Learn - • How to achieve transactional atomicity through database interactivity using ado.net API.
Time	120 Mins

The Tables used in the Lab:

```
/*

 create table accmaster(accno int primary key, accname varchar(30),
                        accbal numeric(10,2) check(accbal >= 5000))

create table acctran(tranno int identity primary key,accno int references accmaster,trtype
varchar(1),
                        tramt numeric(10,2) check(tramt >= 500))

insert into accmaster values(1,'anand',50000)
insert into accmaster values(2,'milind',50000)

 */
```

1. Add a new Windows form; Name it Transaction Demo
   a. Drag 3 Labels, 3 TextBoxes (txtacno, txttrtype, txtamt), 1 Button (btnsave)



2. In the code behind of the form put these namespaces
   using System.Data.SqlClient;


```
using System.Configuration;
```

3. Declare the following 2 objects as form level members

```
SqlConnection nwcon;
SqlTransaction tr;
```

4. Following is the Form Load code:

```
private void TransactionDemo_Load(object sender, EventArgs e)
 {
    string constr = ConfigurationManager.ConnectionStrings
                                        ["labdemoconnectstring"].ConnectionString;
    nwcon = new SqlConnection(constr);
    nwcon.Open();
 }
```

5. Add the following 2 functions into the Form Class:

```
private void UpdateBal(string accno, string trtype, double tranamt)
 {

    SqlCommand cmd_upd = new SqlCommand
              ("update accmaster set accbal = accbal + @amt where accno = @accno",
     nwcon);

    cmd_upd.Transaction = tr;
    cmd_upd.Parameters.Add("@amt", SqlDbType.Decimal);
    cmd_upd.Parameters.Add("@accno", SqlDbType.Int, 4);

    if (trtype.ToLower().Equals("w"))
       cmd_upd.Parameters["@amt"].Value = -1 * tranamt;
    else
       cmd_upd.Parameters["@amt"].Value = tranamt;

    cmd_upd.Parameters["@accno"].Value = accno;

    cmd_upd.ExecuteNonQuery();
 }


private void SaveStatement(string accno, string trtype, double tranamt)
 {
    string constr = ConfigurationManager.ConnectionStrings
                                        ["labdemoconnectstring"].ConnectionString;

    SqlCommand cmd_ins = new SqlCommand
                        ("insert into acctran values(@accno,@trtype,@tramt)", nwcon);
    cmd_ins.Transaction = tr;
    cmd_ins.Parameters.Add("@tramt", SqlDbType.Money, 4);
    cmd_ins.Parameters.Add("@accno", SqlDbType.Int, 4);
```



```
cmd_ins.Parameters.Add("@trtype", SqlDbType.VarChar, 1);

cmd_ins.Parameters["@accno"].Value = accno;
cmd_ins.Parameters["@tramt"].Value = tranamt;
cmd_ins.Parameters["@trtype"].Value = trtype;

cmd_ins.ExecuteNonQuery();
 }
```

6. The Save Button Click Code:

```
private void btnsave_Click(object sender, EventArgs e)
 {
        if (!txttrtype.Text.ToLower().Equals("w") &&
                                        !txttrtype.Text.ToLower().Equals("d"))
        {
           MessageBox.Show("Transaction type should be 'W' or 'D' ");
           return;
        }

        tr = nwcon.BeginTransaction();

        try
        {
            UpdateBal(txtacno.Text, txttrtype.Text, double.Parse(txtamt.Text));
            SaveStatement(txtacno.Text, txttrtype.Text, double.Parse(txtamt.Text));
            MessageBox.Show("Transaction Saved");
            tr.Commit();
        }
        catch (SqlException sqlex)
        {
             tr.Rollback();
             MessageBox.Show(sqlex.Message);
        }
        finally
        {
              tr=null;
        }
}
```

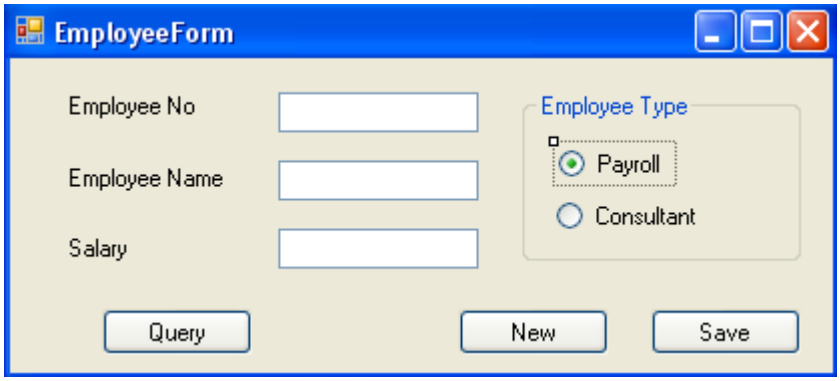
7. Run the Application
   a. Our first inputs Accno:= 1, Amt:= 10000, Tran type:= D Click Save, click Ok on Msgbox
   b. Once it succeeds
   c. Goto Backend (SQL Management Studio for SQL 2005/2008) connect to server and check whether deposit has succeeded
   d. Our Second inputs Accno:= 2, Amt:= 200, Tran type:= D Click Save, it will MsgBox an error

13. Goto Backend and check that deposit has not happened and ATOMICITY is maintained.
Note: No tables are updated neither **AccMaster** NOR **AccTran**

Lab 10. Using New Features of ADO.NET 4.5

Description	In this lab we will be using Employee Table. The end user will experience responsive UI and code will be more secure.
Goals	To Learn - • Asynchronous Processing in ADO.NET 4.5 • Using SqlCredential to Securely Store User Name and Password in a Connection String.
Time	30 Mins

Open the Employee Form designed in Lab 1. The Employee Form should look like this:



Add the following code in the EmployeeForm.cs:

```
private async static Task<SqlDataReader> RetrieveEmployeeRecord(SqlCommand
dbCommand)
{
                return await dbCommand.ExecuteReaderAsync();
}
```

Change the code of EmployeeForm_Load as follows:

```
private async void EmployeeForm_Load(object sender, EventArgs e)
{
                con = new
SqlConnection(@"server=atrgsql\sql2005;database=labdemos;" +
                                                "user id=sqluser;password=sqluser");
                 await con.OpenAsync();
}
```


Change the code of btnquery_Click as follows:

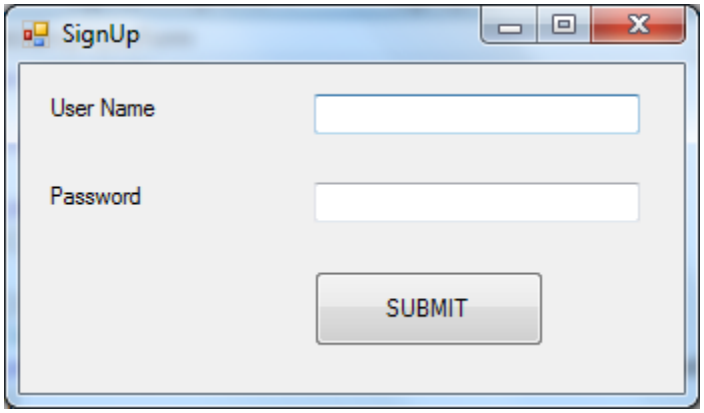
```
private void btnquery_Click(object sender, EventArgs e)
{
      try
       {
          SqlDataReader dreader = null;
          //The Procedure to execute
          SqlCommand cmd = new SqlCommand("GetEmployeeById",con);
           cmd.CommandType = CommandType.StoredProcedure;
          //define procedure parameter
          SqlParameter prm;
          prm = new SqlParameter();
          prm.SqlDbType = SqlDbType.Int;
          prm.Direction = ParameterDirection.Input;
          prm.ParameterName = "@eno";
           cmd.Parameters.Add(prm);
          //assign parameter value
          cmd.Parameters["@eno"].Value = int.Parse(txtempno.Text);
          //execute
          dreader = await RetrieveEmployeeRecord(cmd);
          //if employee record found
           if (dreader.Read())
          {
                    txtempname.Text = dreader["empname"].ToString();
                       txtsalary.Text = dreader["empsal"].ToString();
                    if (dreader["emptype"].ToString() == "P")
                              rdpayroll.Checked = true;
                    else
                              rdconsultant.Checked = true;
          }
          else
          {
                    btnnew_Click(btnnew, e);
                     MessageBox.Show("No such employee");
          }
          dreader.Close();
     }
    catch (SqlException sqlex)
     {
          MessageBox.Show(sqlex.Message);
      }
}
```

Run the Application

a. Click New to clear the form and type in an Employee No and click Query.

Assignment 1

To Do:
Add windows form named SignUp.cs to the project. Design the SignUp form as shown below:



- Remove the username and password from the connection string in EmployeeForm.
- On SignUp form, accept username and password from the user. Accept password as SecureString.
- Modify the code in EmployeeForm to include this username and password in connection string (Hint: Use SqlCredential).
- On EmployeeForm, enter the existing employee id and click on Query button. You should be able to see the details of the employee.

