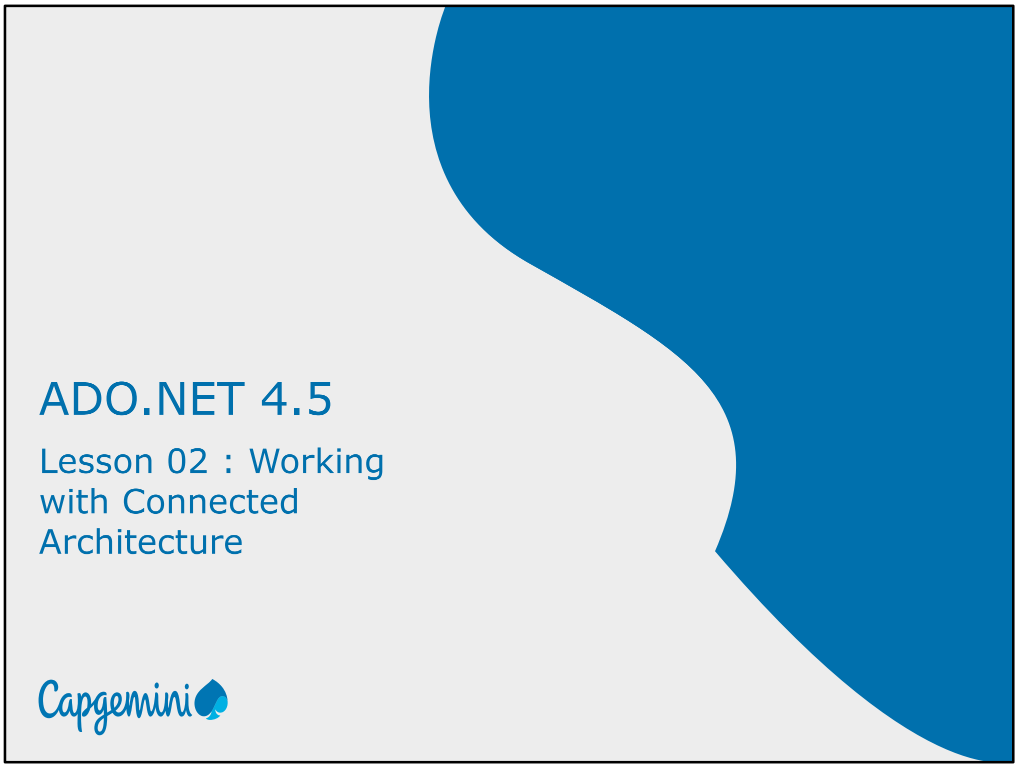
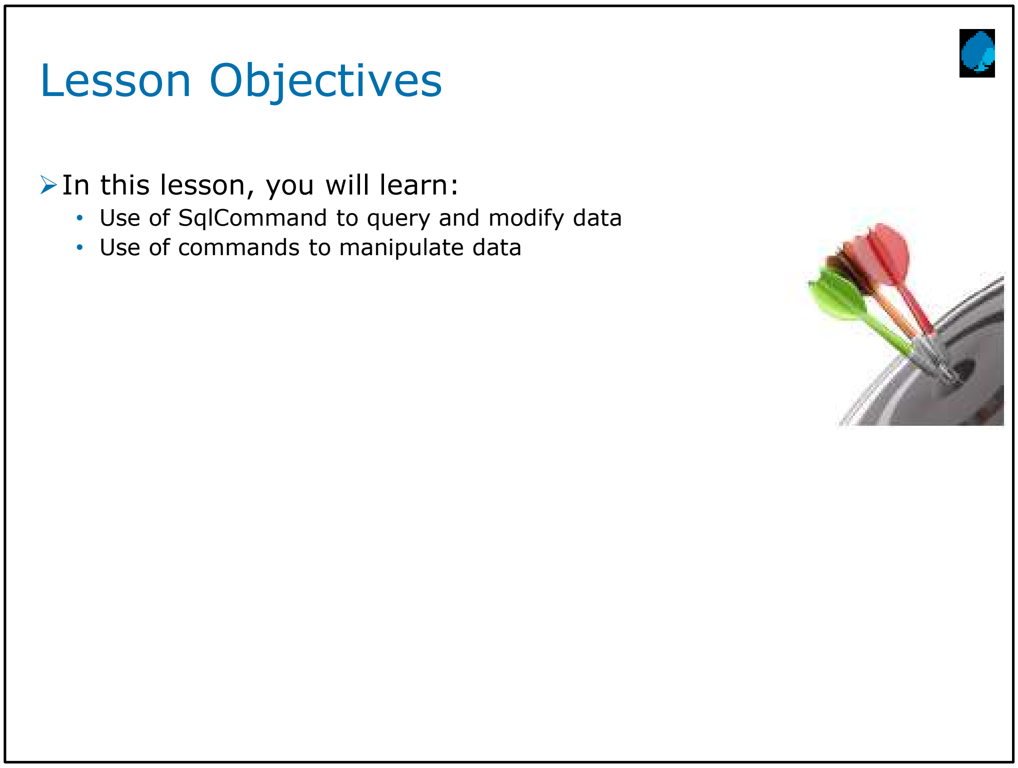


ADO.NET 4.5

Lesson 02 : 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

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.

2.1: Use of Command object in Connected Environment

Creating and Executing Commands

- The Command object represents an executable command on the underlying data source
- It may or may not return any results
- It can be used to manipulate existing data, query existing data, and update or even delete existing data
- The Command object also exposes Parameter collection which can be used for executing parameterized queries

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.

2.1: Use of Command object in Connected Environment

SQLCommand Constructors

- The SQLCommand class exposes four constructors:
  - SQLCommand()
  - SQLCommand(String commandText)
  - SQLCommand(String commandText,SQLConnection con)
  - SQLCommand(String commandText, SQLConnection con, SQLTransaction trans)

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.

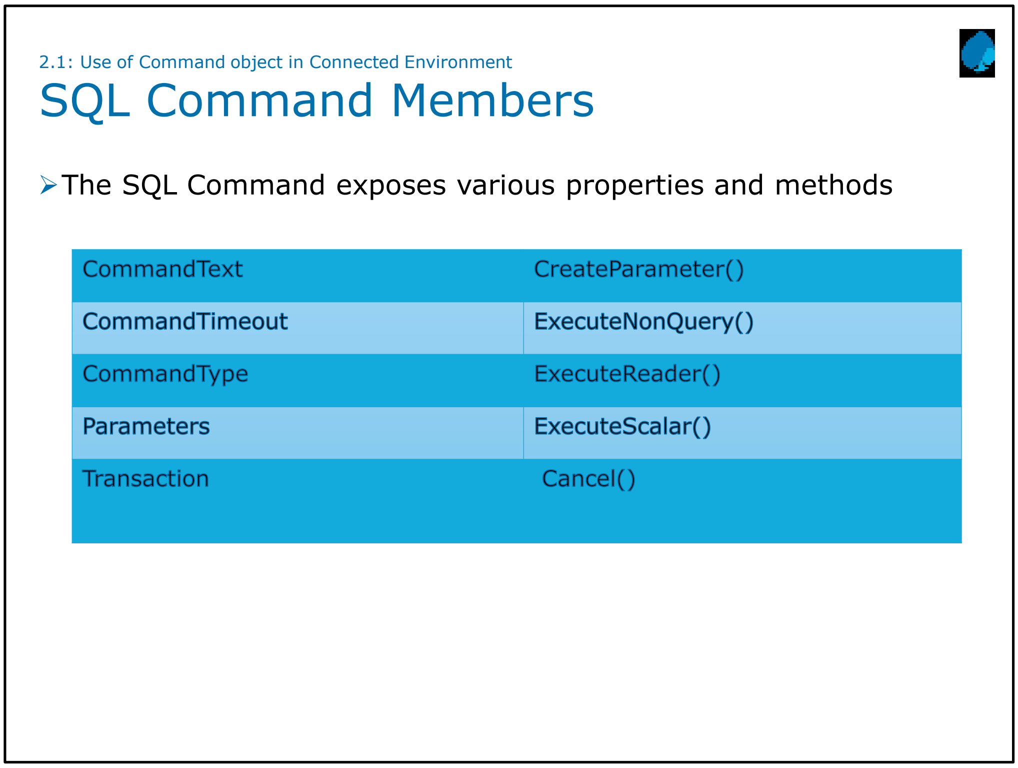
SQLCommand Constructors (Contd.):

```
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();
```



2.1: Use of Command object in Connected Environment

SQL Command Members

➢ The SQL Command exposes various properties and methods

CommandText	CreateParameter()
CommandTimeout	ExecuteNonQuery()
CommandType	ExecuteReader()
Parameters	ExecuteScalar()
Transaction	Cancel()

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.

2.1: Use of Command object in Connected Environment

Sql Data Reader

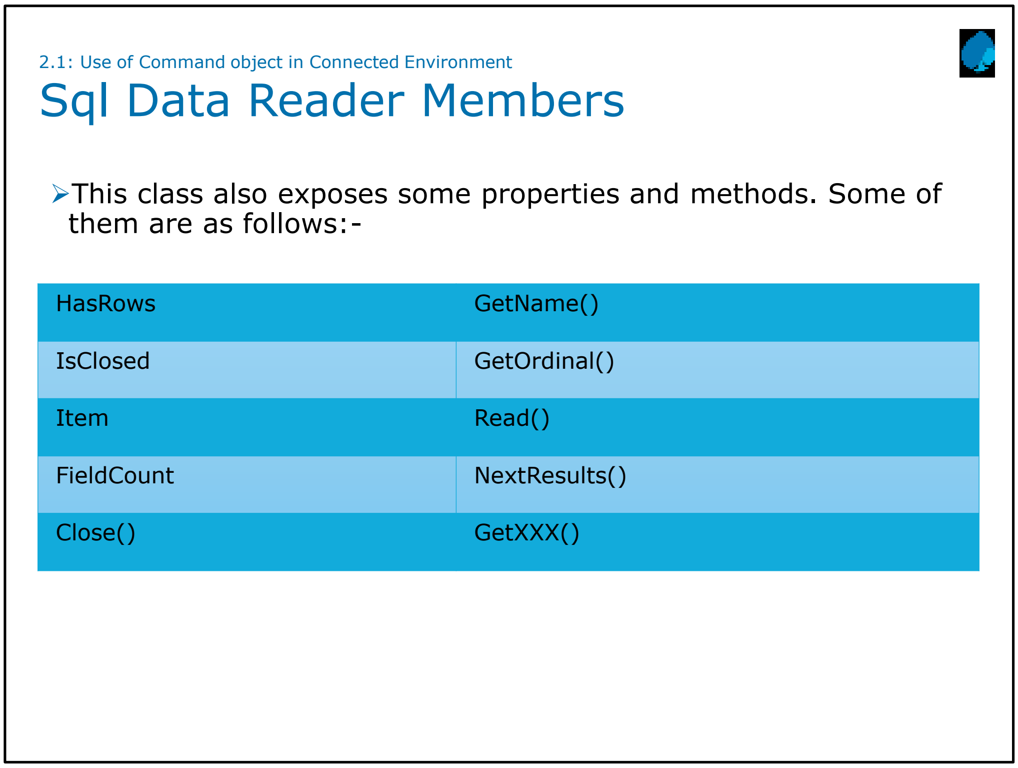
- It is one of the core objects used in ADO.NET connected architecture to store and read data
- You can use the Sql Data Reader object to examine the results of a query one row at a time
- When you move forward to the next row, the contents of the previous row are discarded
- The Sql Data Reader doesn't support updating
- The data returned by the Sql Data Reader is read-only and forward-only

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.



2.1: Use of Command object in Connected Environment

Sql Data Reader Members

- This class also exposes some properties and methods. Some of them are as follows:-

HasRows	GetName()
IsClosed	GetOrdinal()
Item	Read()
FieldCount	NextResults()
Close()	GetXXX()

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

2.1: Use of Command object in Connected Environment

Executing Stored Procedure

- A stored procedure written in SQL server can be called and executed through ADO.NET
- The Sql Command object is used to execute stored procedures
- Example :

```
Sql Command cmd=new SqlCommand("Query_Emp",con);
 cmd.CommandType=CommandType.Stored Procedure;
```

Executing Stored Procedure:

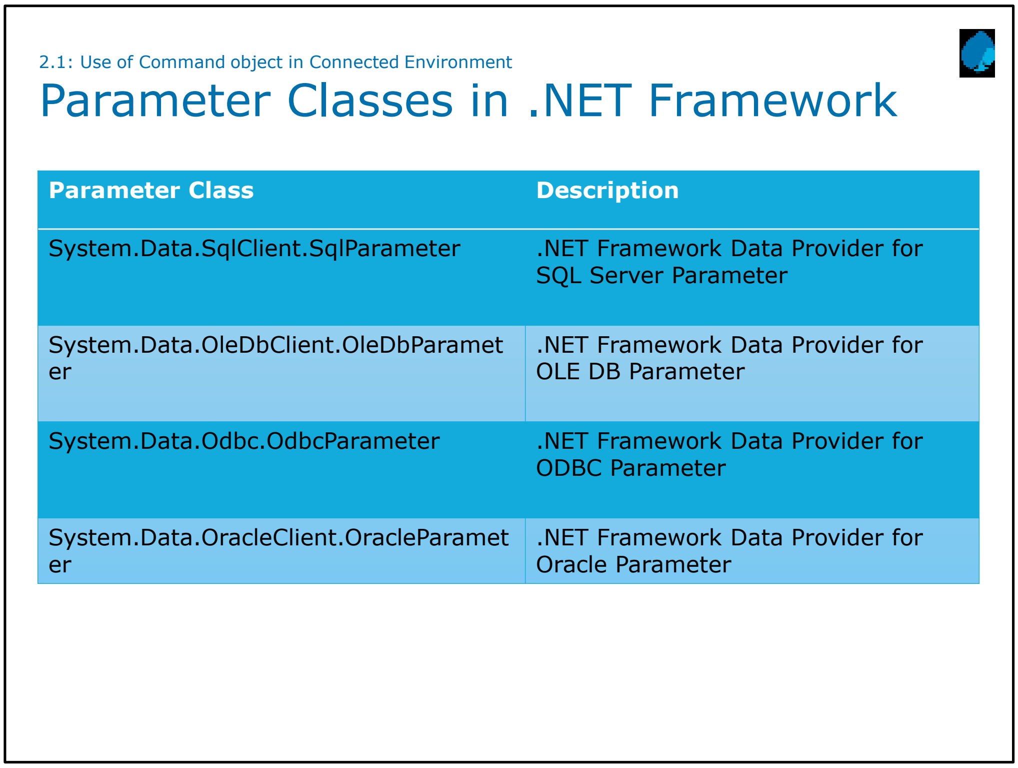
A stored procedure is a reusable sub routine stored in a database. SQL Server compiles stored procedures which makes it more efficient to use. Therefore , rather than dynamically building queries in the code, we can always take advantage of the reusability and performance benefits of stored procedures.
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.

2.1: Use of Command object in Connected Environment

Passing parameters into Command Object

- Every Command object has an associated collection of Parameter objects
- The Parameter object is a provider-specific object
- Sql Command uses a Sql Parameter, an Ole Db Command uses an Ole Db Parameter, and so on
- SQL statements and stored procedures can take input, output, and bidirectional parameters
- Stored procedures can also return a value
- You must configure your command object so that it handles parameters and return values correctly

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.



2.1: Use of Command object in Connected Environment

Parameter Classes in .NET Framework

Parameter Class	Description
System.Data.SqlClient.SqlParameter	.NET Framework Data Provider for SQL Server Parameter
System.Data.OleDbClient.OleDbParamet er	.NET Framework Data Provider for OLE DB Parameter
System.Data.Odbc.OdbcParameter	.NET Framework Data Provider for ODBC Parameter
System.Data.OracleClient.OracleParamet er	.NET Framework Data Provider for Oracle Parameter

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.

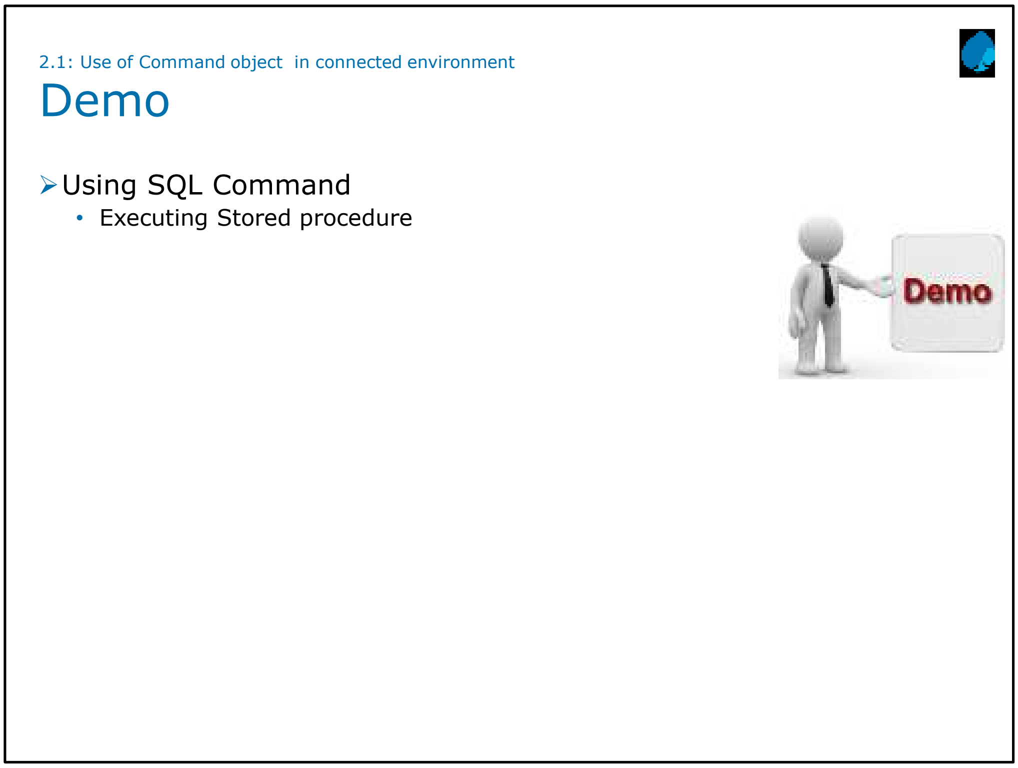
DbType – Set this property to the data type of the parameter.

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.

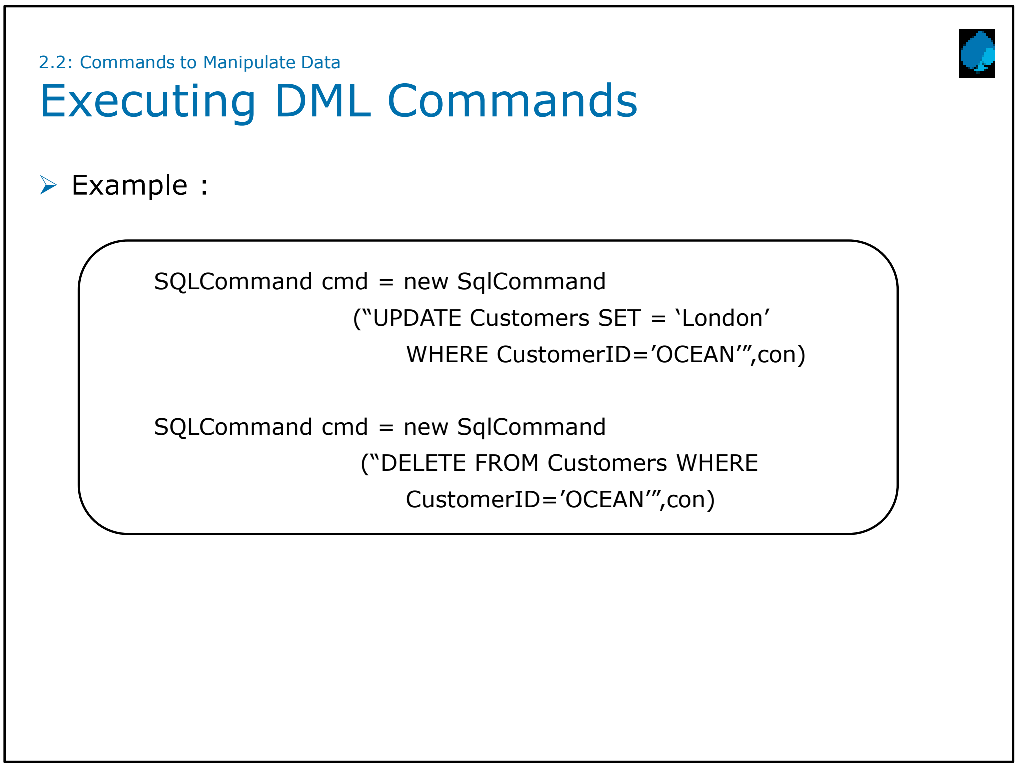
5. Value - For input or bidirectional parameters, set the Value property before you run the command. For output, bidirectional parameters and for stored procedure return value, you can retrieve the Value property after you run the command.



2.1: Use of Command object in connected environment

Demo

- Using SQL Command
  - Executing Stored procedure



2.2: Commands to Manipulate Data

Executing DML Commands

➢ Example :

```
SQLCommand cmd = new SqlCommand
                         ("UPDATE Customers SET = 'London'
                                WHERE CustomerID='OCEAN'",con)

SQLCommand cmd = new SqlCommand
                          ("DELETE FROM Customers WHERE
                                CustomerID='OCEAN'",con)
```

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

2.3: Managing Commands

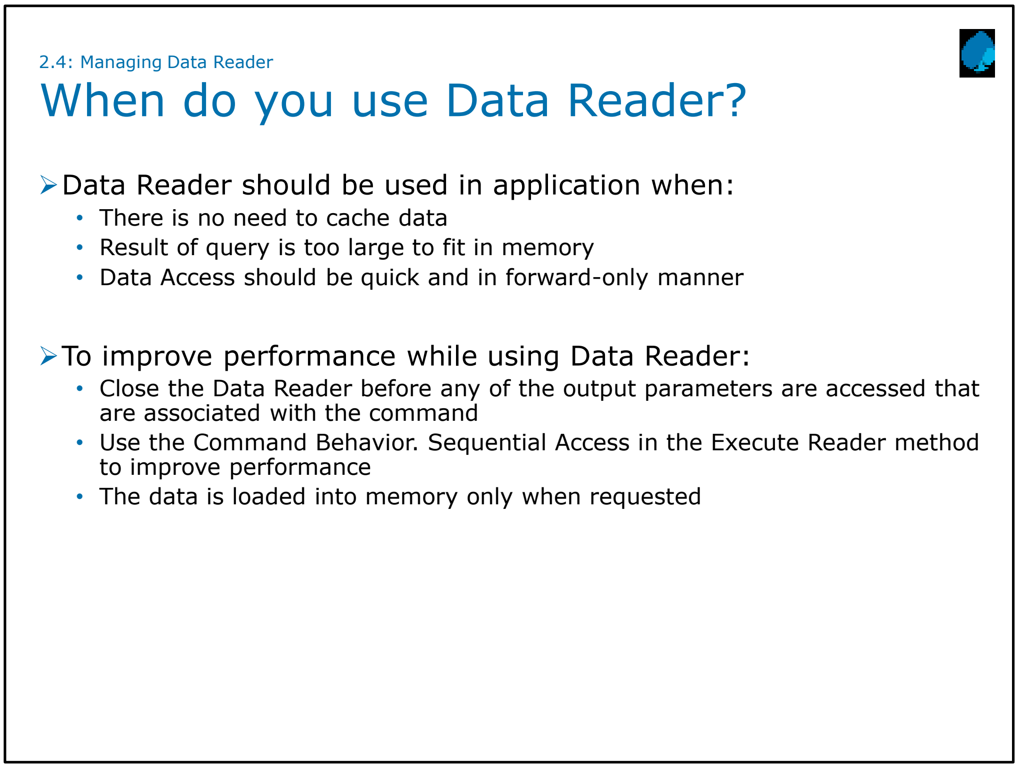
Using Commands

- Specify Command Type while using Stored Procedures
- Many ADO.NET objects refer to metadata information, in such cases, specify schema and metadata explicitly
- Use Execute Scalar and Execute Non Query as applicable
- Test Null Values for any columns allowing Nulls

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)
```



2.4: Managing Data Reader

When do you use Data Reader?

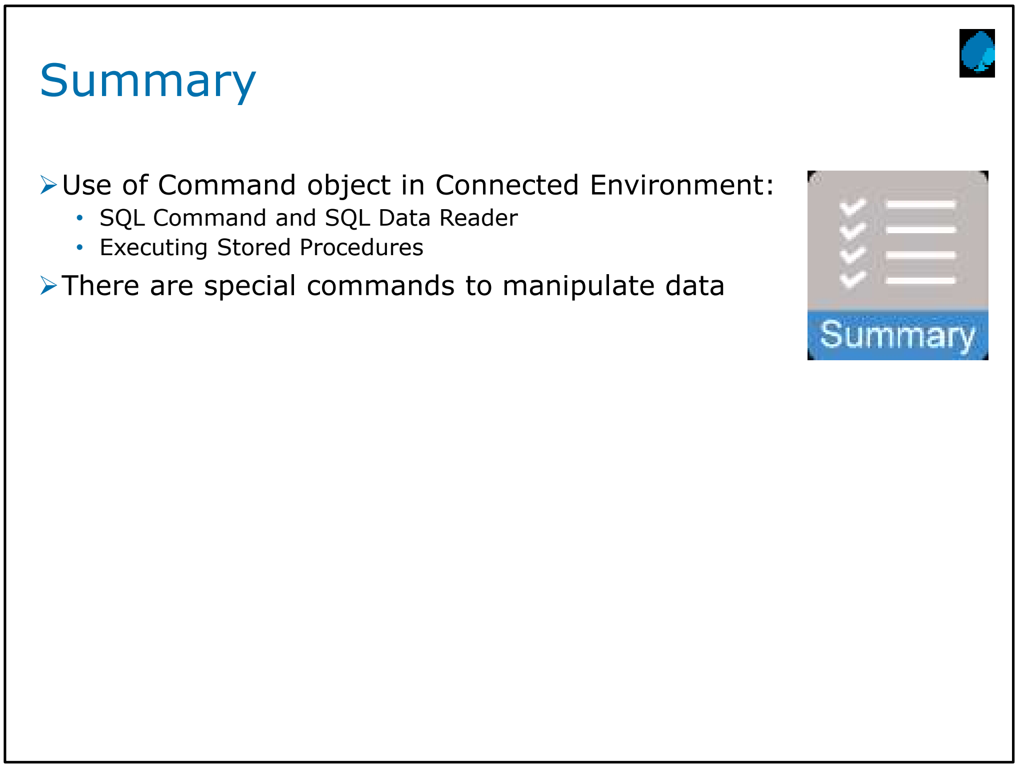
- Data Reader should be used in application when:
  - There is no need to cache data
  - Result of query is too large to fit in memory
  - Data Access should be quick and in forward-only manner
- To improve performance while using Data Reader:
  - Close the Data Reader before any of the output parameters are accessed that are associated with the command
  - Use the Command Behavior. Sequential Access in the Execute Reader method to improve performance
  - The data is loaded into memory only when requested

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.



Summary

- Use of Command object in Connected Environment:
  - SQL Command and SQL Data Reader
  - Executing Stored Procedures
- There are special commands to manipulate data

Summary

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 __________.

Add the notes here.