



# COMPSCI 280 S1 2011 Enterprise Software Development

ADO.NET  
Data Commands and DataReaders



## Agenda & Reading

- ▶ **Agenda:**
  - ▶ Overview of Data Providers
  - ▶ Understanding Data Commands
    - ▶ Creating Data Commands
    - ▶ Executing Data Commands
    - ▶ Parameters
  - ▶ Understanding Data Readers
    - ▶ Understanding Data Readers
    - ▶ Creating Data Readers
    - ▶ Reading from Data Readers
- ▶ **Recommended Reading:**
  - ▶ Microsoft ADO.NET 2.0 step by step, Rebecca M. Riordan
    - ▶ Chapter 3 : Using Data Commands and DataReaders, 45-53, 59-68
  - ▶ Working with Commands
    - ▶ <http://msdn2.microsoft.com/en-us/library/ms254953.aspx>
  - ▶ Retrieving Data Using a DataReader
    - ▶ [http://msdn2.microsoft.com/en-us/library/haa3afyz\(VS.80\).aspx](http://msdn2.microsoft.com/en-us/library/haa3afyz(VS.80).aspx)
- ▶ **Namespaces Covered:**
  - ▶ System.Data.OleDb
- ▶ **Hands-on Lab:**

2

▶ Lecture20Lab: Data Commands & DataReaders

COMPSCI280

Handout20



## Data Types

| Data Type                | Returned as Class | SQL           | Microsoft Access      |
|--------------------------|-------------------|---------------|-----------------------|
| boolean                  | bool              | Bit           | Yes/No                |
| short (2 bytes)          | short             | SMALLINT      | Number (Integer)      |
| integer (4 bytes)        | int               | INT           | Number (Long Integer) |
| float                    | float             | FLOAT         | Number (Single)       |
| double                   | double            | DOUBLE        | Number (Double)       |
| Currency                 | decimal           | Money         | Currency              |
| String (fixed length)    | String            | Char(n)       | Text                  |
| String (variable length) | String            | varchar(n)    | Text                  |
| DateTime                 | DateTime          | Date/DateTime | Date/Time             |

- ▶ char(n): Fixed length character string, with user-specified length n
- ▶ varchar(n): Variable length character strings, with user-specified maximum length n.

3

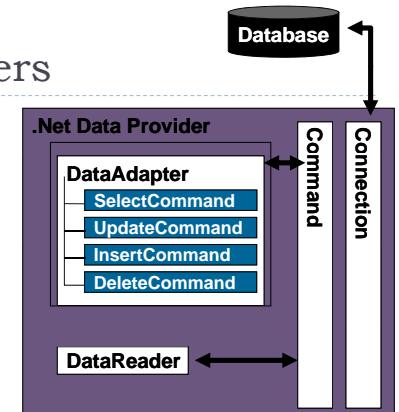
COMPSCI280

Handout20



## Overview of Data Providers

- ▶ **Core elements**
  - ▶ Connection
  - ▶ Data Command
  - ▶ DataReader
  - ▶ DataAdapter/TableAdapter
- ▶ **Two basic ways to work with data**
  - ▶ Using a data command object
    - ▶ Returns single value
    - ▶ Returns multiple values: read by data readers
  - ▶ Using a DataSet (in-memory store of the records)
    - ▶ loads or updates values by using data adapters



- ◆ A data provider in the .NET framework serves as a bridge between an application and a data source.
- ◆ The Connection, Command, DataReader, and DataAdapter objects represent the core elements of the .NET framework data provider model.
- ◆ In the DataSet model, you create an in-memory store of the records you want to work with, load the data using a data adapter, manipulate the data, and then optionally, use the data adapter to write changes back to the database.
- ◆ In the command model, you configure a data command object with an SQL statement or the name of a stored procedure. You then execute the data command.

4

COMPSCI280

Handout20



# Understanding Data Commands

- ▶ **Data Commands**
  - ▶ Reference to an SQL statement or stored procedure that you can execute directly
- ▶ **Properties:**
  - ▶ **CommandType**
    - ▶ **Property:** type of the data Command
      - Text -> A SQL Text command
      - StoredProcedure -> The name of a stored procedure
  - ▶ **CommandText**
    - ▶ String, SQL statement or the name of a stored procedure in the data source
  - ▶ **Connection**
    - ▶ References to the Connection object on which the command will be executed
  - ▶ **Parameter**
    - ▶ A command may require you to pass parameters values along with it (input parameters)

◆ The SQL statement of the Command object can be queried and modified using the CommandText property.  
 ◆ A stored procedure is a piece of code that operates on a database, typically written in SQL.  
 ◆ When using the Command object with a stored procedure, you may set the CommandType property of the Command object to StoredProcedure. With a CommandType of StoredProcedure, you may use the Parameters property of the Command to access input and output parameters and return values. The Parameters property can be accessed regardless of the Execute method called. However, when calling ExecuteReader, return values and output parameters will not be accessible until the DataReader is closed.



# Data Commands (con't)

- ▶ **Constructors:**
  - ▶ OleDbCommand()
  - ▶ OleDbCommand(commandText, connection)
- ▶ **Methods:**
  - ▶ ExecuteScalar
    - ▶ Executes commands that return a scalar value
  - ▶ ExecuteNonQuery
    - ▶ Executes commands to create/edit/remove tables, stored procedures
  - ▶ ExecuteReader
    - ▶ Executes select statements and read with data readers
- ▶ **Creating Data Commands**
  - ▶ Method 1: Creating at Run Time
  - ▶ Method 2: Using the Toolbox

◆ After establishing a connection to a data source, you can execute commands and return results from the data source using a Command object.  
 ◆ You can create a command using the Command constructor, which takes optional arguments of an SQL statement to execute at the data source.  
 ◆ The Command object exposes several Execute methods you can use to perform the intended action. When returning results as a stream of data, use ExecuteReader to return a DataReader object. Use ExecuteScalar to return a singleton value (for example, an aggregate value). Use ExecuteNonQuery to execute commands that do not return rows.



# Creating Data Commands

## ▶ Method 1: Creating at Run Time

- ▶ Using Constructors:
  - ▶ OleDbCommand(), or
  - ▶ OleDbCommand(commandText, connection)

```
OleDbCommand cmdGet = new OleDbCommand();
cmdGet.CommandType = CommandType.StoredProcedure;
cmdGet.CommandText = "[Ten Most Expensive Products]";
cmdGet.Connection = cnNorthwind;
```

Stored Procedure

```
OleDbCommand cmdCount = new OleDbCommand("Select count(*) from Products", cnNorthwind);
```

SQL statement

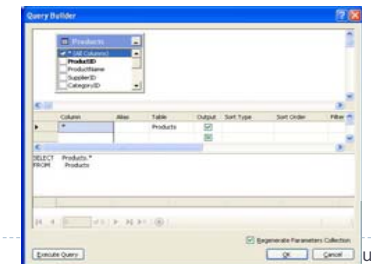
## ▶ Method 2: Using the Toolbox

- ▶ Add a Command to the Toolbox
  - ▶ Open the **Toolbox**, and right-click the **Data** tab
  - ▶ Select **Choose Items** from the Context menu.
  - ▶ Select the **OleDbCommand** check box
- ▶ Add a Data Command to a Form
  - ▶ Drag an **OleDbCommand** onto the form



# Creating Data Commands (con't)

- ▶ To configure the Data Command
  - ▶ Select the Data Command in the component tray
  - ▶ In the properties window, select the Connection property
    - Set the Connection to cnNorthwind
  - ▶ Select CommandText Property:
    - Click the ellipsis button
    - Select Products table from the AddTable dialog box
    - Insert the SQL statements or select the checkbox in the Diagram pane of the query builder
  - ▶ (Demo: 20.1 AddDataCommand.wmv)





# ExecuteScalar

Example: CommandDemo

- ▶ ExecuteScalar
  - ▶ Returns a single value (useful for returning aggregate values)
- ▶ To execute the ExecuteScalar command
  - ▶ Create a new Data Command
  - ▶ Open the connection
  - ▶ Call the command's ExecuteScalar method
  - ▶ Close the connection

```
private void btnConnect_Click(object sender, EventArgs e) {
    ...
    cnNorthwind.Open();
    ...
}

private void btnClose_Click(object sender, EventArgs e) {
    ...
    cnNorthwind.Close();
    ...
}
```

```
string sql = "Select Count(*) from Products";
OleDbCommand cmdCount = new OleDbCommand(sql, cnNorthwind);
MessageBox.Show(cmdCount.ExecuteScalar());
```

Returns 77

- ◆ The ExecuteScalar method returns as a scalar value. For example, you may want to return the result of an aggregate function such as count(\*), Sum(Price), Avg(Quantity).
- ◆ Note: You should use a try/catch-finally block to open the connection, execute the Command and close the connection afterwards. Please refer to our examples for more details.



# ExecuteNonQuery

- ▶ Executes SQL's insert, delete and update statements
  - ▶ Returns the number of rows affected
  - ▶ Returns -1 if operation fails
- ▶ To execute the ExecuteNonQuery command
  - ▶ Create a new Data Command
  - ▶ Open the connection
  - ▶ Call the command's ExecuteNonQuery method
  - ▶ Close the connection

## ▶ Example: Insert SQL statement

```
string sql = "Insert into Shippers values (4, 'ABC', '123')";
OleDbCommand cmdInsert = new OleDbCommand(sql, cnNorthwind);
MessageBox.Show(cmdInsert.ExecuteNonQuery());
```

## ▶ Example: Delete SQL statement

```
string sql = "delete from Shippers where ShipperID=4";
OleDbCommand cmdDelete = new OleDbCommand(sql, cnNorthwind);
MessageBox.Show(cmdDelete.ExecuteNonQuery());
```



# Parameters

- ▶ OleDbParameter
  - ▶ Represents a parameter to a special SELECT SQL OleDbCommand with a WHERE clause is incomplete until user enters value at run time
- ▶ Constructors
  - ▶ OleDbParameter(Name, type)
  - ▶ OleDbParameter(Name, type, size)
    - ▶ Initializes a new instance of the OleDbParameter class that uses the parameter name, data type, and length.
- ▶ Example: Select SQL statement with a parameter
  - ▶ Create a new parameter
  - ▶ Set the value
  - ▶ Add the parameter to the Parameter collection of a Data Command
  - ▶ Execute the data command

```
string sql = "Select Count(*) from Products where CategoryID = ?";
OleDbCommand cmdCount = new OleDbCommand(sql, cnNorthwind);
OleDbParameter param = new OleDbParameter("CategoryID", OleDbType.VarChar);
param.Value = 1;
cmdCount.Parameters.Add(param);
MessageBox.Show(cmdCount.ExecuteScalar());
```

Create a new parameter and set its value

Return 12

- ◆ Note: The OLE DB.NET Framework Data Provider uses positional parameters that are marked with a question mark (?)



# Parameters (con't)

- ▶ Example: Insert SQL statement with parameters
  - ▶ Add the two parameters to the Parameter Collection
    - ▶ Parameters.Add(name, type, size)
  - ▶ Set values of the two parameters
  - ▶ Execute the data command
- ▶ Date types of a field for use in an OleDbParameter.
  - ▶ OleDbType.VarWChar: A variable-length, null-terminated stream of Unicode characters. This maps to String.
  - ▶ OleDbType.Integer: A 32-bit signed integer. This maps to Int32.

```
OleDbCommand cmdAdd = new OleDbCommand("INSERT INTO Shippers (CompanyName, Phone) VALUES (?, ?)",
cnNorthwind);
cmdAdd.Parameters.Add(new OleDbParameter("CompanyName", OleDbType.VarWChar, 50));
cmdAdd.Parameters.Add(new OleDbParameter("Phone", OleDbType.VarWChar, 24));
cmdAdd.Parameters["CompanyName"].Value = "ABC";
cmdAdd.Parameters["Phone"].Value = "(503) 123-4567"

MessageBox.Show(cmdInsert.ExecuteNonQuery());
```

- ◆ The order of the parameters added to the Parameters collection must match the order of the parameters defined in your SQL statement or stored procedure. Therefore, the order in which Parameter objects are added to the Parameters collection must directly correspond to the position of the question mark placeholder for the parameter.
- ◆ 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.



## Parameters (con't)

- ▶ Example: Update SQL statement with parameters
  - ▶ Add the two parameters to the Parameter Collection
    - ▶ Parameters.Add(name, type, size)
  - ▶ Set values of the two parameters
  - ▶ Execute the data command
- ▶ Example: Delete SQL statement with a parameter
  - ▶ Create a new parameter: ShipperID
  - ▶ Set the value
  - ▶ Add the parameter to the Parameter collection of a Data Command
  - ▶ Execute the data command

```
OleDbCommand cmdUpdate = new OleDbCommand("UPDATE Shippers SET CompanyName=? WHERE ShipperID=?",
cnNorthwind);
cmdUpdate.Parameters.Add(new OleDbParameter("CompanyName", OleDbType.VarChar, 50));
cmdUpdate.Parameters.Add(new OleDbParameter("ShipperID", OleDbType.Integer));
cmdUpdate.Parameters["ShipperID"].Value = 5;
cmdUpdate.Parameters["CompanyName"].Value = "DEF";

MessageBox.Show(cmdInsert.ExecuteNonQuery());

OleDbCommand cmdDelete = new OleDbCommand("DELETE FROM Shippers WHERE ShipperID=?", cnNorthwind);
cmdDelete.Parameters.Add(new OleDbParameter("ShipperID", OleDbType.Integer));
cmdDelete.Parameters["ShipperID"].Value = "5";

MessageBox.Show(cmdDelete.ExecuteNonQuery());
```

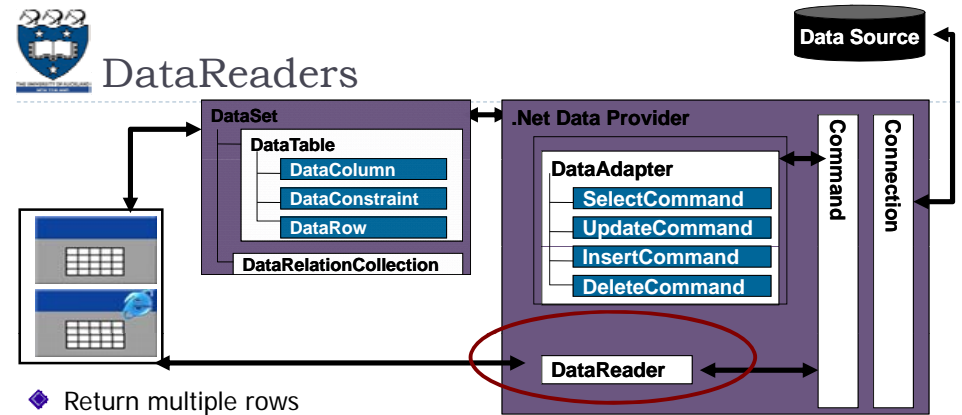
13

COMPSCI280

Handout20



## DataReaders



- ◆ Return multiple rows
- ◆ Efficiently retrieve a forward-only stream of data
- ◆ Used in scenarios where data need not be updateable

◆ You can use the ADO.NET DataReader to retrieve a read-only, forward-only stream of data from a database. Results are returned as the query executes, and are stored in the network buffer on the client until you request them using the Read method of the DataReader. Using the DataReader can increase application performance both by retrieving data as soon as it is available, rather than waiting for the entire results of the query to be returned, and (by default) storing only one row at a time in memory reducing system overhead.

14

COMPSCI280

Handout20



## DataReaders (con't)

- ▶ Properties
  - ▶ HasRow: Gets a value that indicates whether the OleDbDataReader contains one or more rows
  - ▶ Item: Gets the value of a column in its native format.
  - ▶ FieldCount: Gets the number of columns in the current row.
- ▶ Methods
  - ▶ Get<Type>: returns the values using Getxxx method where xxx is the type of the data to be retrieved
  - ▶ GetName: Gets the name of the specified column.
  - ▶ GetOrdinal: Gets the column ordinal, given the name of the column.
  - ▶ Read: Moves to the next row and returns true if it exists
  - ▶ Close: Closes the OleDbDataReader object.
- ▶ To create a DataReader
  - ▶ Open an connection
  - ▶ Execute the ExecuteReader method of a Data Command

```
cnNorthwind.Open();
OleDbDataReader rdrCustomers = cmdPrint.ExecuteReader();
```

15

COMPSCI280

Handout20



## Data Readers

- ▶ To retrieve data
  - ▶ Use the HasRow method to check if the OleDbDataReader contains one or more rows
  - ▶ Use a loop to read data from the DataReader
    - ▶ GetString(0), GetInt32(1)
      - return the values using Getxxx method where xxx is the type of the data to be retrieved (zero-based)
    - ▶ Item[int] or Item[String]
      - returns the values using the Column's ordinal or its name
  - ▶ Close the DataReader and the connection

Use indexer in C#  
E.g. rdrCustomers[0]

```
if (rdrCustomers.HasRows) {
    while (rdrCustomers.Read()) {
        txtOutput.AppendText(rdrCustomers.GetString(0) + " " + rdrCustomers["UnitPrice"] +
            Environment.NewLine);
    }
}
...
rdrCustomers.close()
```

Column name

Get a string from the first column

- ◆ You can use the Read method of the DataReader object to obtain a row from the results of the query. You can access each column of the returned row by passing the name or ordinal reference of the column to the DataReader. However, for best performance, the DataReader provides a series of methods that allow you to access column values in their native data types (GetDateTime, GetDouble, GetInt32, and so on). Using the typed accessor methods, when the underlying data type is known, reduces the amount of type conversion required when retrieving the column value.
- ◆ Note that while a DataReader is open, the Connection is use exclusively by that DataReader. You will not be able to execute any commands for the Connection, until the original DataReader is closed.

16

COMPSCI280

Handout20



# Summary

Example: ReaderDemo



- ▶ Summary
  - ▶ Understanding Data Commands
    - ▶ Creating Data Commands
    - ▶ Executing Data Commands
    - ▶ Parameters
  - ▶ Understanding Data Readers
    - ▶ Creating Data Readers
  - ▶ Advantages
    - ▶ More control over execution
      - Get more direct control over how and when an SQL statement or stored procedure is executed and what becomes of the results or return values
    - ▶ Less Overhead
      - Bypass storing data in a DataSet (DataSet requires memory)
      - Reduce some overhead in your application
  - ▶ Recommendations for accessing data using DataReaders
    - ▶ When you use the data only once
    - ▶ Useful in Web Forms (require less programming)
    - ▶ Getting scalar value from the database
    - ▶ Performing a non-query operation
    - ▶ Getting read-only data to display in a form