VB.NET & SQL Server

Base de Dados - 2015/16
Carlos Costa

(Book: Mastering Microsoft Visual Basic 2010)

(Stream versus Set) Data Access

- Framework we use to access databases is known as ADO.NET (ADO stands for Active Data Objects) and it provides two basic methods of accessing data:
  - *stream-based data access**, which establishes a stream to the database and retrieves the data from the server
    - you must create the appropriate INSERT/UPDATE/DELETE statements and then execute them against the database.
    - stream-based approach relies on the DataReader object, which makes the data returned by the database available to your application.
  - *set-based data access*, which creates a special data structure at the client and fills it with data.
    - DataSet
      - contains one or more DataTable objects
        - » made up of DataRow objects
    - set-based approach uses the same objects as the stream-based approach behind the scenes, and it abstracts most of the grunt work required to set up a link to the database, retrieve the data, and store it in the client computer's memory

* used in this presentation

Basic Data-Access Classes

The cycle of a data-driven application:

1. Retrieve data from the database.

2. Present data to the user.

3. Allow the user to edit the data.

4. Submit changes to the database.

Retrieve data from the database

Classes:

- **Connection**
  - channel between your application and the database

- **Command**
  - execute the command (SQL Statement) against the database

- **DataReader**
  - allows you to read the data returned by the selection query, one row at a time

The Connection Class

```
-- Connection Class Usage

Imports System.Data.SqlClient

Dim CN As New SqlConnection("Data Source = localhost;" &
"Initial Catalog = Northwind; uid = user_name;" &
"password = user_password")


CN.Open()

...

CN.Close()
```

Connection - Example

```
-- Example: Test SQL Server Connection

Imports System.Data.SqlClient

Private Sub TestDBConnection(ByVal dbServer As String, ByVal dbName As String,
                             ByVal userName As String, ByVal userPass As String)

  Dim CN As New SqlConnection("Data Source = " + dbServer + " ;" +
                              "Initial Catalog = " + dbName + "; uid = " + userName + ";" +
                              "password = " + userPass)

  Try
    CN.Open()
    If CN.State = ConnectionState.Open Then
      MsgBox("Successful connection to database " & CN.Database & " on the " & CN.DataSource &
                                    " server", MsgBoxStyle.OkOnly, "Connection Test")
    End If

  Catch ex As Exception
    MsgBox("FAILED TO OPEN CONNECTION TO DATABASE DUE TO THE FOLLOWING ERROR" & vbCrLf &
                              ex.Message, MsgBoxStyle.Critical, "Connection Test")
  End Try

  If CN.State = ConnectionState.Open Then
    CN.Close()

End Sub
```

The Command Class

- To execute a SQL statement against a database, you must initialize a Command object and set its Connection property to the appropriate Connection object.

  Dim CMD As New **SqlCommand**(<SQL Statement>, <CN>)

- Command object simply submits a SQL statement to the database and retrieves the results.

- Several methods available:

  - **ExecuteReader**: used with SELECT statements that return rows from one or more tables.

  - **ExecuteNonQuery**: executes INSERT/DELETE/UPDATE statements that do not return any rows, just an integer value, which is the number of rows affected by the query.

  - **ExecuteScalar**: returns a single value, which is usually the result of an aggregate operation, such as the count of rows meeting some criteria, the sum or average of a column over a number of rows, and so on.

ExecuteNonQuery – Example

-- Example: Execute SQL Update Statement

```vbnet
Imports System.Data.SqlClient

Private Sub TestCommandUpdate(ByVal CN As SqlConnection)

  Dim CMD As New SqlCommand( "UPDATE Products SET UnitPrice = UnitPrice * 1.07 " &

                                      "WHERE CategoryID = 3",
CN)

  CN.Open()

  Dim rows As Integer
  rows = CMD.ExecuteNonQuery

  If rows = 1 Then
    MsgBox("Table Products updated successfully")
  Else
    MsgBox("Failed to update the Products table") End If

  If CN.State = ConnectionState.Open Then
    CN.Close()

End Sub
```

ExecuteScalar – Example

```
-- Example: Execute SQL Select Statement – returns a scalar value

Imports System.Data.SqlClient

Private Sub TestCommandSelectScalar(ByVal  CN As SqlConnection)

  Dim CMD As New SqlCommand( "SELECT COUNT(*) FROM Customers", CN)

  CN.Open()

  Dim count As Integer
  count = CMD.ExecuteScalar

  MsgBox(”Number of Customers: " + count)

  If CN.State = ConnectionState.Open  Then
    CN.Close()

End Sub
```

ExecuteReader – Example

```
-- Example: Execute SQL Select Statement - returns a record set
Imports System.Data.SqlClient

Private Sub TestCommandSelectRecordSet (ByVal CN As SqlConnection)

  Dim CMD As New SqlCommand( "SELECT * FROM Customers", CN)

  CN.Open()

  Dim reader As SqlDataReader
  reader = CMD.ExecuteReader

  While reader.Read
       ' process the current row in the result set
  End While

  If CN.State = ConnectionState.Open  Then
    CN.Close()

End Sub
```

Command – Non SQL Statement

- Command object is not always a SQL statement: could be the name of a stored procedure, or the name of a table, in which case it retrieves all the rows of the table.

- You can specify the type of statement you want to execute with the CommandType property:
  - Text (for SQL statements),
  - StoredProcedure (for stored procedures),
  - TableDirect (for a table).

```
-- Stored Procedure [Ten Most Expensive Products]
Dim CMD As New SqlCommand
CMD.Connection = CN
CMD.CommandText = "[Ten Most Expensive Products]"
CMD.CommandType = CommandType.StoredProcedure
```

```
-- Customers table
Dim CMD As New SqlCommand
CMD.Connection = CN
CMD.CommandText = "Customers"
CMD.CommandType = CommandType.TableDirect
```

The DataReader Class

- SELECT statements, retrieve a set of rows from one or more joined tables, the *result set.*

- ExecuteReader method, which returns a DataReader object — a SqlDataReader object.

- The DataReader class provides the members for reading the results of the query in a forward-only manner.

  – Read method: reads and advances the current pointer to the next row in the result set.

    • Item property: read the individual columns of the current row

DataReader – Example 1

```vbnet
-- Example: Execute SQL Select Statement - returns a record set

Imports System.Data.SqlClient

Private Sub TestCommandSelectDataReader (ByVal CN As SqlConnection)
  Dim CMD As New SqlCommand( "SELECT LastName + ' ' + FirstName AS Name, " &
                             "Title, Extension, HomePhone FROM Employees", CN)

  CN.Open()
  Dim reader As SqlDataReader
  reader = CMD.ExecuteReader
  Dim str As String = ""
  Dim count As Integer = 0

  While reader.Read
    str &= Convert.ToString(reader.Item("Name"))  & vbTab
    str &= Convert.ToString(reader.Item("Title"))  & vbTab
    str &= Convert.ToString(reader.Item("Extension"))  & vbTab
    str &= Convert.ToString(reader.Item("HomePhone"))  & vbTab
    str &= vbCrLf
    count += 1
  End While

  MsgBox("Read " & count.ToString & " rows: " & vbCrLf & str)

  If CN.State = ConnectionState.Open  Then CN.Close()
End Sub
```

DataReader – Example 2

```vb
-- Example: Execute SQL Select Statement with WHERE - returns a record

Imports System.Data.SqlClient

Private Sub TesTSelectDataReader (ByVal CN As SqlConnection, ByVal ssn as String)

  Dim querySQL As String = "SELECT * FROM Employee WHERE Ssn='" + ssn + "'"
  Dim CMD As New SqlCommand(querySQL , CN)

  CN.Open()
  Dim reader As SqlDataReader
  reader = CMD.ExecuteReader
  Dim str As String = ""

  If reader.Read Then
    str &= Convert.ToString(reader.Item("Fname"))  & vbTab
    str &= Convert.ToString(reader.Item("Lname"))  & vbTab
    str &= Convert.ToString(reader.Item("Address"))  & vbTab
    str &= vbCrLf
  End If

  MsgBox("Employee: " & vbCrLf & str)

  If CN.State = ConnectionState.Open  Then CN.Close()
End Sub
```

Commands with Parameters

- Most SQL statements and stored procedures accept parameters, and you should pass values for each parameter before executing the query.

  SELECT * FROM Customers WHERE Country = @country

  @country parameter must be set to a value

- Command object exposes the Parameters property.

- Must set up a Parameter object for each parameter; set its name, type, and value; and then add the Parameter object to the Parameters collection of the Command object.

Commands with Parameters - Example

```
-- Example: Several Possibilities
```

```vbnet
Dim Command As New SqlCommand
Command.CommandText = "SELECT * FROM Customers WHERE Country = @country"
Command.Parameters.Add("@country", SqlDbType.VarChar, 15)      -- Define type
Command.Parameters("@country").Value = "Italy"                 -- Set value
```

```vbnet
Dim param As New SqlParameter("@country", SqlDbType.VarChar, 15)
param.Value = "Italy"
CMD.Parameters.Add param
```

```vbnet
CMD.Parameters.Add("@country", SqlDbType.VarChar, 15).Value = "Italy"
```

```vbnet
CMD.Parameters.Add("@country", "Italy")

-- the type of the query or stored procedure's argument, and it's resolved at runtime.
```

Retrieving Multiple Values from a Stored Procedure

- Another property of the Parameter class is the Direction property, which determines whether the stored procedure can alter the value of the parameter.

  - ParameterDirection:
    - Input - parameter is used to pass information to the procedure
    - Output - parameter is used to pass information back to the calling application
    - InputOutput - parameter is used to pass information to the procedure, as well as to pass information back to the calling application
    - ReturnValue - return a value

Example 1/3 – The Scenario

- A stored procedure (SP) returns the total of all orders, as well as the total number of items ordered by a specific customer.

- SP accepts as a parameter the ID of a customer, obviously, and it returns two values: the total of all orders placed by the specified customer and the number of items ordered.

- To make the stored procedure a little more interesting, we'll add a return value, which will be the number of orders placed by the customer.

Example 2/3 – SQL Server

```
-- Example: Creating SP in SQL Server
-- @customerTotal and @customerItems variables are output parameters
-- @customerOrders variable is the procedure's return value
```

```sql
CREATE PROCEDURE CustomerTotals @customerID varchar(5),
                                @customerTotal money OUTPUT,
                                @customerItems int OUTPUT
AS
SELECT @customerTotal = SUM(UnitPrice * Quantity * (1 - Discount))
FROM [Order Details] INNER JOIN Orders ON [Order Details].OrderID =
Orders.OrderID WHERE Orders.CustomerID = @customerID

SELECT @customerItems = SUM(Quantity)
FROM [Order Details] INNER JOIN Orders ON [Order Details].OrderID =
Orders.OrderID WHERE Orders.CustomerID = @customerID

DECLARE @customerOrders int

SELECT @customerOrders = COUNT(*)
FROM Orders
WHERE Orders.CustomerID = @customerID

RETURN @customerOrders
```

Example 3/3 – VB.NET

-- Example: Calling the SP in VB.NET

```vbnet
Private Sub ExecSP(ByVal customerID As String)
  Dim CMD As New SqlCommand
  CMD.Connection = CN
  CMD.CommandText = "CustomerTotals"
  CMD.CommandType  = CommandType.StoredProcedure
  CMD.Parameters.Add("@customerID", SqlDbType.VarChar, 5).Value = customerID

  CMD.Parameters.Add("@customerTotal", SqlDbType.Money)
  CMD.Parameters("@customerTotal").Direction = ParameterDirection.Output

  CMD.Parameters.Add("@customerItems", SqlDbType.Int)
  CMD.Parameters("@customerItems").Direction = ParameterDirection.Output

  CMD.Parameters.Add("@orders", SqlDbType.Int)
  CMD.Parameters("@orders").Direction = ParameterDirection.ReturnValue
  CN.Open()
  CMD.ExecuteNonQuery()
  CN.Close()

  Dim items As Integer = Convert.ToInt32(CMD.Parameters("@customerItems").Value)
  Dim orders As Integer = Convert.ToInt32(CMD.Parameters("@orders").Value)
  Dim ordersTotal As Decimal = Convert.ToDouble(CMD.Parameters("@customerTotal").Value)
  MsgBox("Customer BLAUS has placed " & orders.ToString & " orders " & "totaling $" &
        Math.Round(ordersTotal, 2).ToString("#,###.00") & " and " & items.ToString & " items")
End Sub
```

Data Reader - Properties and methods 1/2

- HasRows
  - This is a Boolean property that specifies whether there's a result set to read data from. If the query selected no rows at all, the HasRows property will return False.

- FieldCount
  - This property returns the number of columns in the current result set. Note that the DataReader object doesn't know the number of rows returned by the query. Because it reads the rows in a forward-only fashion, you must iterate through the entire result set to find out the number of rows returned by the query.

- Read
  - This method moves the pointer in front of the next row in the result set. Use this method to read the rows of the result set, usually from within a While loop.

- Get<type>
  - There are many versions of the Get method with different names, depending on the type of column you want to read. To read a Decimal value, use the GetDecimal method; to retrieve a string, use the GetString method; to retrieve an integer, call one of the GetInt16, GetInt32, or GetSqlInt64 methods; and so on. To specify the column you want to read, use an integer index value that represents the column's ordinal, such as Reader.GetString(2). The index of the first column in the result set is zero.

- GetSql<type>
  - There are many versions of the GetSql method with different names, depending on the SQL type of the column you want to read. To read a Decimal value, use the GetSqlDecimal method; to retrieve a string, use the GetSqlString method; to retrieve an integer, call one of the GetSqlInt16, GetSqlInt32, or GetSqlInt64 methods; and so on. To specify the column you want to read, use an integer index value that represents the column's ordinal, such as Reader.GetSqlString(2). The index of the first column in the result set is zero.

Data Reader - Properties and methods 2/2

- GetValue
  - If you can't be sure about the type of a column, use the GetValue method, which returns a value of the Object type. This method accepts as an argument the ordinal of the column you want to read.

- GetValues
  - This method reads all the columns of the current row and stores them into an array of objects, which is passed to the method as an argument. This method returns an integer value, which is the number of columns read from the current row.

- GetName
  - Use this method to retrieve the name of a column, which must be specified by its order in the result set. To retrieve the name of the first column, use the expression Reader.GetName(0). The column's name in the result set is the original column name, unless the SELECT statement used an alias to return a column with a different name.

- GetOrdinal
  - This is the counterpart of the GetName method, and it returns the ordinal of a specific column from its name. To retrieve the ordinal of the CompanyName column, use the expression Reader.GetName("CompanyName").

- IsDbNull
  - This method returns True if the column specified by its ordinal in the current row is null. If you attempt to assign a null column to a variable, a runtime exception will be thrown, so you should use this method to determine whether a column has a value and handle the null values from within your code.

CLR Types (.NET) versus SQL Types

The Get<Type> methods return data types recognized by the Common Language Runtime (CLR), whereas the GetSql<Type> methods return data types recognized by SQL Server. There's a one-to-one correspondence between most types but not always. In most cases, we use the Get<Type> methods and store the values in VB variables, but you may want to store the value of a field in its native format. Use the SQL data types only if you're planning to move the data into another database. For normal processing, you should read them with the Get<type> methods, which return CLR data types recognized by VB. The following table summarizes the CLR and SQL data types:
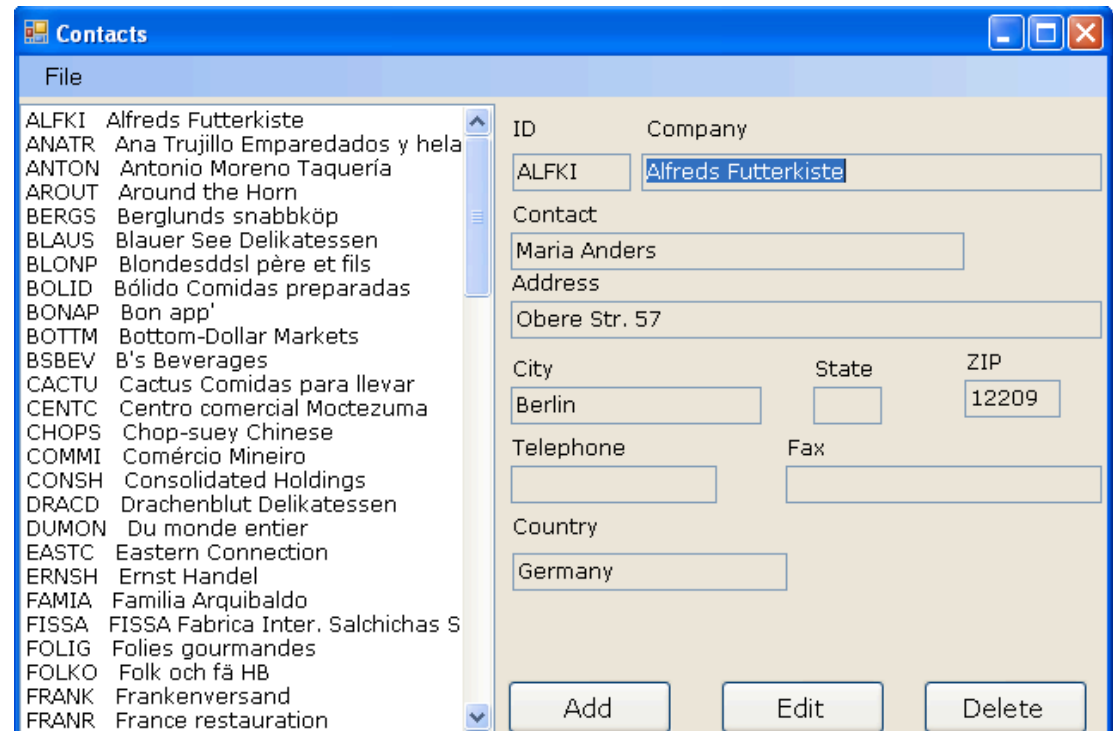
CLR DATA TYPE	SQL DATA TYPE
Byte	SqlByte
Byte()	SqlBytes
Char()	SqlChars
DateTime	SqlDateTime
Decimal	SqlDecimal
Double	SqlDouble
	SqlMoney
Single	SqlSingle
String	SqlString
	SqlXml

T-SQL + VB

in action

Simple Application

- Using the Microsoft Northwind database

- Objective:

  1. Display list of contacts

  2. Add new contact

  3. Edit contact

  4. Delete contact

Code available…

VB.NET Contact Class

-- VB Contact Class to represent a database Customer record

```vb
Public Class Contact
    Private _customerID As String
    Private _companyName As String
    Private _contactName As String
    Private _address1 As String
    Private _address2 As String
    Private _city As String
    Private _state As String
    Private _zip As String
    Private _country As String
    Private _tel As String
    Private _fax As String

    Property CustomerID As String
        Get
            Return _customerID
        End Get
        Set(ByVal value As String)
            _customerID = value
        End Set
    End Property

    ...

END CLASS
```

Loading the Customers table to ListBox

-- Fill VB ListBox1 object with Customers record-set

```vb
Private Sub LoadToolStripMenuItem_Click(…) Handles LoadToolStripMenuItem.Click

  CMD.CommandText = "SELECT * FROM Customers"
  CN.Open()

  Dim RDR As SqlDataReader
  RDR = CMD.ExecuteReader

  ListBox1.Items.Clear()

  While RDR.Read
    Dim C As New Contact
    C.CustomerID = RDR.Item("CustomerID")
    C.CompanyName = RDR.Item("CompanyName")
    C.ContactName = Convert.ToString(IIf(RDR.IsDBNull(RDR.GetOrdinal("ContactName")), "", RDR.Item("ContactName")))
    C.Address1 = Convert.ToString(IIf(RDR.IsDBNull(RDR.GetOrdinal("Address")), "", RDR.Item("Address")))
    C.City = Convert.ToString(IIf(RDR.IsDBNull(RDR.GetOrdinal("City")), "", RDR.Item("City")))
    C.State = Convert.ToString(IIf(RDR.IsDBNull(RDR.GetOrdinal("Region")), "", RDR.Item("Region")))
    C.ZIP = Convert.ToString(IIf(RDR.IsDBNull(RDR.GetOrdinal("PostalCode")), "", RDR.Item("PostalCode")))
    C.Country = Convert.ToString(IIf(RDR.IsDBNull(RDR.GetOrdinal("Country")), "", RDR.Item("Country")))
    ListBox1.Items.Add(C)
  End While

  CN.Close()
  currentContact = 0
  ShowContact()

End Sub
```

Adding a new row to the Customers table

```vb
-- Insert a Contact (VB Object) into SQL Server Customers table

Private Sub SubmitContact(ByVal C As Contact)
    CMD.CommandText = "INSERT Customers (CustomerID, CompanyName, ContactName, Address, " & _
                      "City, Region, PostalCode, Country) " & _
                      "VALUES (@CustomerID, @CompanyName, @ContactName, @Address, " & _
                      "@City, @Region, @PostalCode, @Country) "
    CMD.Parameters.Clear()
    CMD.Parameters.AddWithValue("@CustomerID", C.CustomerID)
    CMD.Parameters.AddWithValue("@CompanyName", C.CompanyName)
    CMD.Parameters.AddWithValue("@ContactName", C.ContactName)
    CMD.Parameters.AddWithValue("@Address", C.Address1)
    CMD.Parameters.AddWithValue("@City", C.City)
    CMD.Parameters.AddWithValue("@Region", C.State)
    CMD.Parameters.AddWithValue("@PostalCode", C.ZIP)
    CMD.Parameters.AddWithValue("@Country", C.ZIP)

    CN.Open()

    Try
        CMD.ExecuteNonQuery()
    Catch ex As Exception
        Throw New Exception("Failed to add contact. " & vbCrLf & "ERROR MESSAGE: " & vbCrLf & ex.Message)
    Finally
        CN.Close()
    End Try

End Sub
```

Updating a row in the Customers table

```vb
-- Update a Contact (VB Object) in SQL Server Customers table
Private Sub SubmitContact(ByVal C As Contact)
  CMD.CommandText = "UPDATE Customers " & _
          "SET CompanyName = @CompanyName, " & _
          "    ContactName = @ContactName, " & _
          "    Address = @Address, " & _
          "    City = @City, " & _
          "    Region = @Region, " & _
          "    PostalCode = PostalCode, " & _
          "    Country = @Country " & _
          "WHERE CustomerID = @CustomerID"
  CMD.Parameters.Clear()
  CMD.Parameters.AddWithValue("@CustomerID", C.CustomerID)
  CMD.Parameters.AddWithValue("@CompanyName", C.CompanyName)
  CMD.Parameters.AddWithValue("@ContactName", C.ContactName)
  CMD.Parameters.AddWithValue("@Address", C.Address1)
  CMD.Parameters.AddWithValue("@City", C.City)
  CMD.Parameters.AddWithValue("@Region", C.State)
  CMD.Parameters.AddWithValue("@PostalCode", C.ZIP)
  CMD.Parameters.AddWithValue("@Country", C.ZIP)
  CN.Open()
  Try
    CMD.ExecuteNonQuery()
  Catch ex As Exception
    Throw New Exception("Failed to update contact. " & vbCrLf & "ERROR MESSAGE: " & vbCrLf & ex.Message)
  Finally
    CN.Close()
  End Try
End Sub
```

Removing a row from the Customers table

```
-- Delete a Contact (using ContactID attribute) from SQL Server Customers table
Private Sub RemoveContact(ByVal ContactID As String)
    CMD.CommandText = "DELETE Customers WHERE CustomerID=@contactID "
    CMD.Parameters.Clear()
    CMD.Parameters.AddWithValue("@contactID", ContactID)

    CN.Open()

    Try
        CMD.ExecuteNonQuery()
    Catch ex As Exception
        Throw New Exception("Failed to delete contact. " & vbCrLf & "ERROR MESSAGE: " & vbCrLf
& ex.Message)
    Finally
        CN.Close()
    End Try

End Sub
```