

Practical 9: Accessing Database Files

NOTE: Turn on Option Explicit and Option Strict compilation options for your project. Prepare your practical solution **AT HOME**. Practical sessions are for you to present and improve your solution.


Create a new windows form application project. Complete the following requirements:

1. Database

- Add a service-based database file named [DB.mdf] to the project. Set its [Copy to Output Directory] property to [Do not copy].

NOTE: Refer to slide Chapter 9(B) Extra for steps required to create database, table, field, entity classes, etc.

- From menu bar, select [View > Server Explorer] or press [Ctrl + Alt + S] to show the **Server Explorer** window.
- Add a database table named [Customer] with the following fields:

	Name	Data Type	Allow Nulls	Default
	Id	char(4)	<input type="checkbox"/>	
	Name	varchar(30)	<input type="checkbox"/>	
	Gender	varchar(6)	<input type="checkbox"/>	

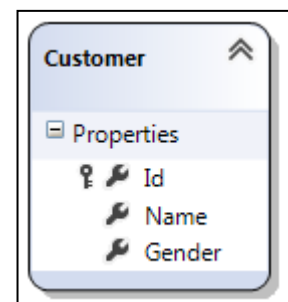
NOTE: For simplification, we use full words for gender (i.e. Female and Male) rather than codes here (i.e. F and M).

- Add the following records to the [Customer] table:

	Id	Name	Gender
	A001	Bae Suzy	Female
	A002	Im Yoona	Female
	A003	Kim Soohyun	Male
	A004	Lee Jongsuk	Male

2. Entity Classes

- Add a LINQ-to-SQL classes file named [DB.dbml] to the project.
- Drag-and-drop the [Customer] table from **Server Explorer** to the **OR Designer**. The [Customer] entity class should be generated.



NOTE: The [DBDataContext] class and [Customer] entity class are generated automatically. We can then communicate with the database using these classes (without writing SQL).

3. Updating Database Location

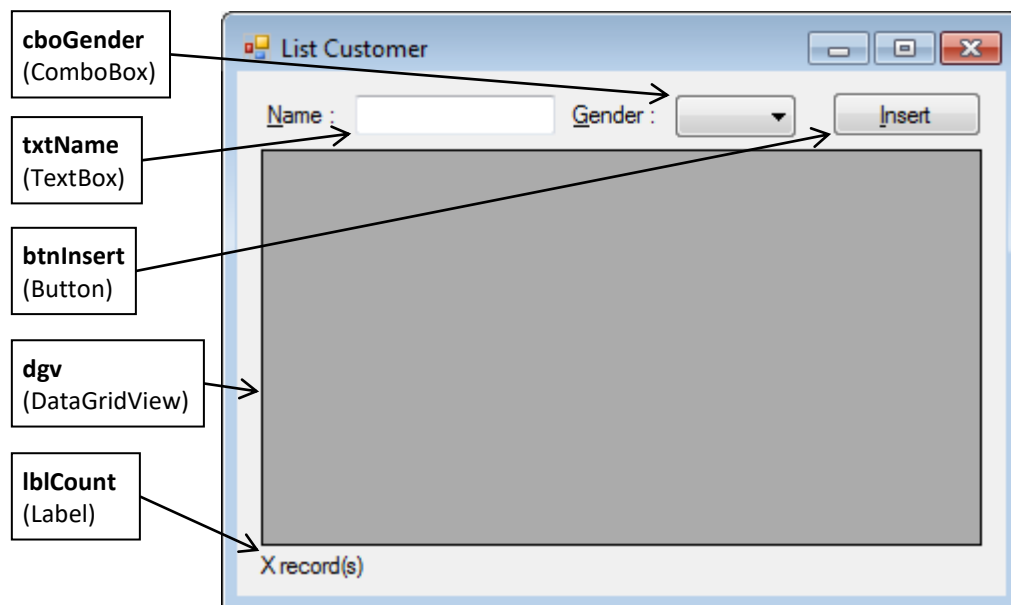
- In **Solution Explorer**, double-click [My Project] to open the **Project Properties** window.
- Under the [Applications] tab, click [View Application Events] button to generate the [ApplicationEvent.vb] file.
- Within the [MyApplication] partial class, add the following event handler for the application **Startup** event:

```
Private Sub MyApplication_Startup(sender As Object,  
                                e As ApplicationServices.StartupEventArgs) _  
    Handles Me.Startup  
    Dim path As String = AppDomain.CurrentDomain.BaseDirectory.Replace("bin\Debug\", "")  
    AppDomain.CurrentDomain.SetData("DataDirectory", path)  
End Sub
```

NOTE: Since we instruct VS not to copy the database to the output directory (i.e. the [bin\Debug] folder), hence we need to update the database location so that our program will look for the database in the project root folder.

4. Form [FrmList]

- Add a form named [FrmList.vb] to the project. The form has the following layout:



FrmList: GUI layout

- Set control properties as follows:

Control	Property	Value
txtName	MaxLength	30
cboGender	DropDownStyle	DropDownList
	Items	All Female Male
btnInsert	Anchor	Top, Right
dgv	AllowUserToAddRows	False
	AllowUserToDeleteRows	False
	Anchor	Top, Bottom, Left, Right
	AutoSizeColumnsMode	AllCells
	AutoSizeRowsMode	AllCells
	ReadOnly	True
	SelectionMode	FullRowSelect

NOTE: To find out more about a particular property, select (i.e. highlight) the property in the **Properties** window and press [F1] (this brings you to the online documentation).

- Configure the form with appropriate usability features: access keys, tab order, accept button, cancel button, etc.
- Create a sub procedure named **BindData()**. It retrieves and displays all [**Customer**] records in [**dgv**] via data binding. Your LINQ query should have a **Where** clause that filters the records by name as entered in [**txtName**] and by gender as selected in [**cboGender**]. It should also update [**lblCount**] with the record count.

```
Private Sub BindData()
    Dim name As String = txtName.Text
    Dim gender As String = cboGender.Text

    Dim db As New DBDataContext()
    Dim rs = ... ← Complete the LINQ query

    dgv.DataSource = rs

    lblCount.Text = rs.Count().ToString("0 record(s)")
End Sub
```

NOTE: Refer to the demo project of **Chapter 9(B) Extra** if you really do not know how.

- Handle the **Load** event of the form: Select the "All" item in [**cboGender**], and call the **BindData()** sub procedure.
- Handle the **TextChanged** event of [**txtName**]: Call the **BindData()** sub procedure.

- Handle the **SelectedIndexChanged** event of [**cboGender**]: Call the **BindData()** sub procedure.
- Test your solution:

The screenshot shows a Windows form titled 'List Customer'. It has a text box for 'Name' and a dropdown for 'Gender' set to 'All'. An 'Insert' button is to the right. Below is a data grid with 4 records. The status bar at the bottom says '4 record(s)'.

	Id	Name	Gender
▶	A001	Bae Suzy	Female
	A002	Im Yoona	Female
	A003	Kim Soohyun	Male
	A004	Lee Jongsuk	Male

FrmList: As the form is loaded, all records are displayed by default (i.e. no filtering)

The screenshot shows the same 'List Customer' form. The 'Name' text box now contains 'o' and the 'Gender' dropdown is set to 'Male'. The data grid now only shows 2 records. The status bar at the bottom says '2 record(s)'.

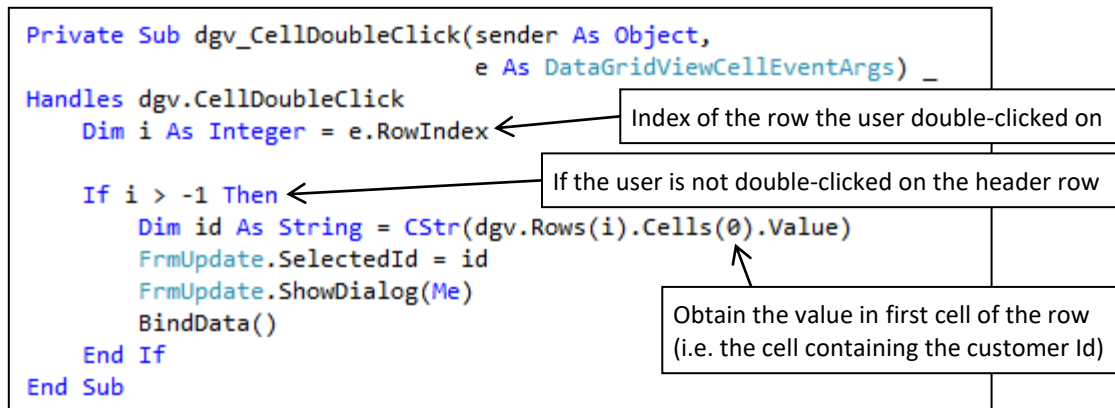
	Id	Name	Gender
▶	A003	Kim Soohyun	Male
	A004	Lee Jongsuk	Male

FrmList: Records are filtered as user are typing in [**txtName**] or selecting an item in [**cboGender**]

- Handle the **Click** event of [**btnInsert**]: Show the form [**FrmInsert**] as a modal dialog, and call the **BindData()** sub procedure after that.

NOTE: Add and program the form [**FrmInsert**] first before doing this.

- Handle the **CellDoubleClick** event of [**dgv**]: Retrieve the customer Id (i.e. cell with index 0) of the row the user double-clicked on, assign the customer Id to the public variable [**SelectedId**] of [**FrmUpdate**], show the form [**FrmUpdate**] as a modal dialog, and call the **BindData()** sub procedure after that. Do all these only if the row index is greater than -1 (i.e. the user is not double-clicked on the header row).



NOTE: Add and program the form [FrmUpdate] first before doing this.

5. Form [FrmInsert]

- Add a form named [FrmInsert.vb] to the project. The form has the following layout:

mskId (MaskedTextBox)

txtName (TextBox)

radFemale and radMale (RadioButton)

btnInsert and btnClose (Button)

FrmInsert: GUI layout

- Set form and control properties as follows:

Control	Property	Value
FrmInsert	AutoValidate	EnableAllowFocusChange
mskId	Mask	>L000
txtName	MaxLength	30
btnClose	CausesValidation	False

- Configure the form with appropriate usability features: access keys, tab order, accept button, cancel button, etc.
- Add an ErrorProvider control to the form. Name it as [err]. Set its [BlinkStyle] property to [NeverBlink].
- Create a sub procedure named **ResetForm()**. It clears the text of [mskId] and [txtName], checks [radFemale], focuses on [mskId], and clears the errors of the ErrorProvider [err].

```
Private Sub ResetForm()
    mskId.Text = ""
    txtName.Text = ""
    radFemale.Checked = True
    mskId.Focus()
    err.Clear()
End Sub
```

- Create a function named **IsDuplicatedId()**. It accepts a customer Id as input parameter. If the customer Id is found in the [**Customer**] table (i.e. duplicated), it returns **True**. Otherwise, it returns **False**.

```
Private Function IsDuplicatedId(id As String) As Boolean
    Dim db As New DBDataContext()
    Return ... ←
End Function
```

Complete the
LINQ query

- Handle the **Shown** event of the form: Call the **ResetForm()** sub procedure.

NOTE: If we are handling the **Load** event, we will unable to set the focus on [**mskId**] as the form is loaded or shown.

- Handle the **Validating** event of [**mskId**]: Perform the necessary validations to ensure the mask is completed, and the customer Id is not duplicated. Set error messages to the ErrorProvider [**err**]. If the customer Id is valid, remove the error.

```
Private Sub mskId_Validating(sender As Object,
                             e As System.ComponentModel.CancelEventArgs) _
    Handles mskId.Validating
    Dim id As String = If(mskId.MaskCompleted, mskId.Text, "")

    If id = "" Then
        err.SetError(mskId, "Please enter valid [Id]")
        e.Cancel = True
    ElseIf IsDuplicatedId(id) Then
        err.SetError(mskId, "[Id] is duplicated")
        e.Cancel = True
    Else
        err.SetError(mskId, Nothing)
    End If
End Sub
```

Set error message to
the ErrorProvider [**err**]

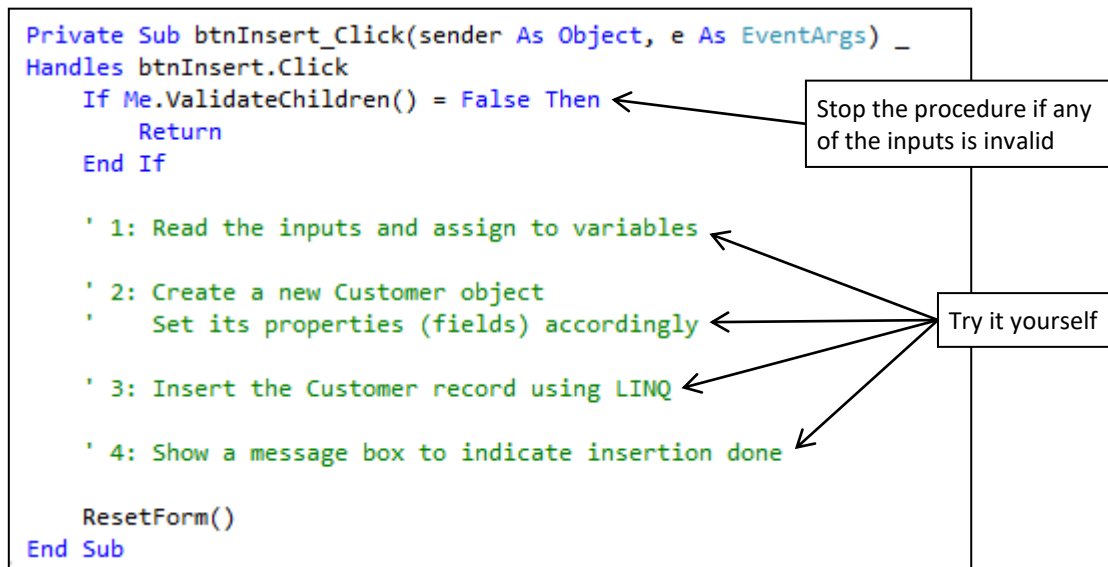
Indicating the validating event
failed (i.e. input is invalid)

Remove the error from
the ErrorProvider [**err**]

- Handle the **Validating** event of [**txtName**]: Perform the necessary validation to ensure the name is not empty. Set error message to the ErrorProvider [**err**]. If the name is valid, remove the error.

NOTE: Try it yourself.

- Handle the **Click** event of [**btnInsert**]: If there is no input error, insert the [**Customer**] record to the database. After that, show a message box to indicate the record has been inserted, and finally reset the form (so that the user can continue to insert another record).



- Handle the **Click** event of [**btnClose**]: Close the form.
- Test your solution:

frmInsert: Errors shown as you tab through the controls with invalid inputs, and [**btnInsert**] has no effect if there are input errors

frmInsert: Error indicators disappear if there are no input errors. Insertion works, and message box tells that the record has been inserted

FrmList: Once [FrmInsert] is closed, the DataGridView will be updated with newly inserted records

Newly inserted customer record

6. Form [FrmUpdate]

- Add a form named [FrmUpdate.vb] to the project. The form has the following layout:

FrmUpdate: GUI layout

- Set form and control properties as follows:

Control	Property	Value
FrmUpdate	AutoValidate	EnableAllowFocusChange
txtName	MaxLength	30
btnDelete	CausesValidation	False
btnClose	CausesValidation	False

- Configure the form with appropriate usability features: access keys, tab order, accept button, cancel button, etc.
- Add an ErrorProvider control to the form. Name it as [err]. Set its [BlinkStyle] property to [NeverBlink].
- Create a class-level public variable named [SelectedId] that allows [FrmList] to communicate the customer Id (of the record to be updated or deleted) to [FrmUpdate].


```
Public SelectedId As String
```

- Create a sub procedure named **ResetForm()**. It clears the text of [**lblId**] and [**txtName**], checks [**radFemale**], focuses on [**txtName**], and clears the errors of the ErrorProvider [**err**].
- Handle the **Shown** event of the form: Call the **ResetForm()** sub procedure. After that, select and display the details of the targeted [**Customer**] record (based on the value of the public variable [**SelectedId**]) in the relevant controls.

```
Private Sub FrmUpdate_Shown(sender As Object, e As EventArgs) Handles MyBase.Shown
    ResetForm()

    Dim db As New DBDataContext()
    Dim c As Customer = db.Customers.FirstOrDefault(Function(o) o.Id = SelectedId)

    If c Is Nothing Then
        MessageBox.Show("Customer not found", "Error",
            MessageBoxButtons.OK, MessageBoxIcon.Error)
        Me.Close()
        Return
    End If

    ' Customer found --> Display details
    lblId.Text = c.Id
    txtName.Text = c.Name

    Select Case c.Gender
        Case "Female" : radFemale.Checked = True
        Case "Male" : radMale.Checked = True
    End Select
End Sub
```

Select the [**Customer**] record that matches the Id

No record is returned (typically, this will not happen)

So, we show an error message, close the form, and return immediately

If we reach here, means everything alright. Hence, display the customer details

Check the right gender

- Handle the **Validating** event of [**txtName**]: Perform the necessary validation to ensure the name is not empty. Set error message to the ErrorProvider [**err**]. If the name is valid, remove the error.
- Handle the **Click** event of [**btnUpdate**]: If there is no input error, update the [**Customer**] record to the database. After that, show a message box to indicate the record has been updated.

```
Private Sub btnUpdate_Click(sender As Object, e As EventArgs) _
    Handles btnUpdate.Click
    If Me.ValidateChildren() = False Then
        Return
    End If

    Dim db As New DBDataContext()
    Dim c As Customer = db.Customers.FirstOrDefault(Function(o) o.Id = SelectedId)
```

Stop, if there are errors

It is necessary to select the [**Customer**] object first before we update it (LINQ requires this)

(Continued next page)

```

If c Is Nothing Then
    MessageBox.Show("Customer not found", "Error",
        MessageBoxButtons.OK, MessageBoxIcon.Error)
    Me.Close()
    Return
End If

' 1: Set the Customer object's properties accordingly
' 2: Update the Customer record using LINQ
' 3: Show a message box to indicate updating done
End Sub

```

Annotations:

- Stop, if the record is not found (points to `Return`)
- Try it yourself (points to lines 1, 2, and 3)

- Handle the **Click** event of [**btnDelete**]: Delete the [**Customer**] record from the database. After that, show a message box to indicate the record has been updated, and close the form.

```

Private Sub btnDelete_Click(sender As Object, e As EventArgs) _
    Handles btnDelete.Click
    Dim db As New DBDataContext()
    Dim c As Customer = db.Customers.FirstOrDefault(Function(o) o.Id = SelectedId)

    If c Is Nothing Then
        MessageBox.Show("Customer not found", "Error",
            MessageBoxButtons.OK, MessageBoxIcon.Error)
        Me.Close()
        Return
    End If

    ' 1: Delete the Customer record using LINQ
    ' 2: Show a message box to indicate deletion done

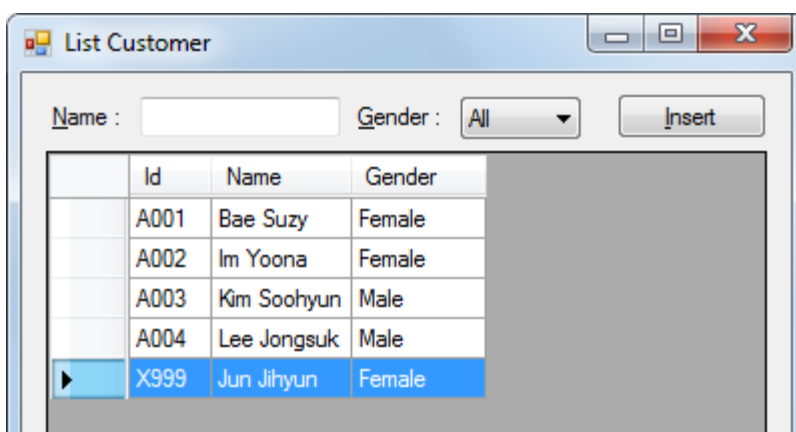
    Me.Close()
End Sub

```

Annotations:

- It is necessary to select the [**Customer**] object first before we delete it (LINQ requires this) (points to `Dim c As Customer = db.Customers.FirstOrDefault(Function(o) o.Id = SelectedId)`)
- Try it yourself (points to lines 1 and 2)
- Close the form. As the record has been deleted, no point to stay open (points to `Me.Close()`)

- Handle the **Click** event of [**btnClose**]: Close the form.
- Test your solution:



FrmList: Double-click on the record you wish to update or delete

Update Customer

Id : X999


Name : Jun Jihyun

Gender : ☒ Female ☐ Male

FrmUpdate: The record will be loaded and displayed in [FrmUpdate]

Update Customer

Id : X999

Name : 

Gender : ☒ Female ☐ Male

FrmUpdate: Trying to update the record with empty name, the input error will be detected


Update Customer

Id : X999

Name : Somebody

Gender : ☒ Female ☐ Male

Update

 Customer [X999] updated


List Customer

Name : Gender : All

	Id	Name	Gender
	A001	Bae Suzy	Female
	A002	Im Yoona	Female
	A003	Kim Soohyun	Male
	A004	Lee Jongsuk	Male
▶	X999	Somebody	Female

FrmUpdate: After a successful update, the message box will be shown. Once the form is closed, the DataGridView in [FrmList] will display the updated value

Delete

 Customer [X999] deleted

FrmUpdate: Deleting the record causes the form to be close immediately, and the DataGridView in [FrmList] will be updated too

List Customer

Name : Gender : All

	Id	Name	Gender
▶	A001	Bae Suzy	Female
	A002	Im Yoona	Female
	A003	Kim Soohyun	Male
	A004	Lee Jongsuk	Male