MZ:2008:4

Mission Report

From a short-term mission on

Visual Basic 2005 Express in connection with Data modelling and SQL and some NADABAS

7-10 July 2008

TA for the Scandinavian Support Program to Strengthen the Institutional Capacity of the National Statistics, Mozambique

Søren Netterstrøm

Instituto Nacional de Estatística

Ref. No.

Table of contents

Executive summary

The primary purpose of this mission was to give training in the use of Visual Basic .NET for building applications against an SQL Server database.

This was carried out as a set of morning sessions, where the principles of Object Orientation, that is the core of VB .NET, was covered and then a simple DB application was build to demonstrate the capacities of VB .NET.

This course can only serve as an introduction to VB .Net. To really exploit any tool like VB .NET, requires practicing over some time gradually building more complex application and learning those components that are needed on the way. Developing application in a system like VB .NET will normally involve a significant amount of self training, using the help facilities of VB .NET, relevant study material as available or searching the Internet for relevant information and then doing some experiments. Then gradually you may build of your own toolbox.

The secondary purpose of the mission was to give assistance on the use of NADABAS. Some minor problems were resolved and a minor change to NADABAS was introduced for more smooth operations.

Appendix 1:

TERMS OF REFERENCE

for a short-term mission on

Visual Basic 2005 Express in connection with Data modelling and SQL and some NADABAS

7 – 11 July, 2008

within the Scandinavian Assistance to Strengthen the Institutional Capacity of INE/Mozambique

Consultant: Søren Netterstrøm
Counterparts: Tomas Bernardo, Anastácia Judas Honwana and more staff at DISI.
Plus also during some time Monica Magaua and others at DCNIG/CN.

Background

It was originally the plan for INE to develop a Data Warehouse system based on 3 database components: A Micro Data Warehouse, A Macro Data Warehouse and a Dissemination Database.

The Dissemination Database is already up and running based on the Scandinavian PC-Axis / PX-Web platform, also used by FAO.

The Data Warehouse strategy and a road map for implementation is described in greater detail in a *Short Term Mission on Data Modelling 31 January - 4 February 2005* by Søren Netterstrøm, MZ:2005:08. The mission then was directed at Anastacia Honwana, Clara Panguana, the developer group at DISI and the LTA on IT Karsten Bormann. The report by Lars Thygesen a *Short Term Mission It Management and Strategic IT use* from September 2006, MZ:2006:10, recommends increased focus on the Data Warehouse. However a need for a more practical / hands approach and training in the related subjects of Data modeling and working with Databases is recognized by INE (DISI) and therefore a mission on Data modeling and SQL was discussed and requested during the visit by Lars Erik Gewalli in November 2006. A mission on this theme was also done 19/2 – 1/3, 2007, MZ:2007:03. The now proposed mission can be seen as a direct continuation of the previous ones and has the advantage of being directly preceded by a mission on Data Modelling and SQL. The mission will also, if necessary, spend up to one day on following up previous missions on NADABAS, the last one described in MZ:2007:10.

A full database system managed with a general language as VB will give INE opportunities to store and backup data in a more efficient manner and to assign different data access rights do different people inside INE.

Also the “though” data management discipline which is a part of full scale database systems will help improve the data quality of INE’s surveys. Further more an increased used of databases will allow INE to develop more ad hoc Client – Server applications.

As a low cost but high tech introduction to database technology is suggested to use the Microsoft MS-SQL 2005 and Visual Basic 2005 in the Express versions. These are provided free of charge by Microsoft and has all the functionality of Microsoft's commercial versions. There are limitations in the amounts of data that it is possible to store in Express version. However for training and familiarization purposes the Express version is more than sufficient.

In order to provide INE with a practical skill building it is planed to build a series of missions around one or two development cases. Each case should lead to a working database.

SDMX is based on XML and XSLT is recommended that training is also provided in this field to INE staff before the end of the year. CPI and National Account data are the data most likely to be requested by international organizations in SDMX-ML. INE may therefore like to enter data from these two subject areas in to a Micro / Macro Data Warehouse model before the end of 2007.

Drawing on Visual Basic and XML it should then be possible for INE to construct a Web service with data in the SDMX-ML format.

Also the LTA on IT left INE by the end of August 2007. Instead it was planned to have a series of short term mission providing INE with gap filling, reflections, discussions and second opinions in the area of IT. This mission should be seen in this context.

Objective

The objective of this mission is to strengthen the practical and theoretical knowledge of data handling at INE, through the use of real data from the Consumer Price Index as a case for the work with VB 2005 Express on a database in MS-SQL 2005 Express.

To demonstrate basic functions in the VB language to extract, manipulate and present data from the database.

The mission is strictly related to the previous SQL mission and will, through hands on training, show how data is entered and extracted to a database from other file formats.

Expected results

- A revision of the MS Visual Basic 2005 Express Installation at a number of workstations for training purposes
- An introduction/repetition of basic OOP concepts like inheritance, polymorphism e encapsulation as well as objects, classes, etc.
- An introduction/repetition of basic VB concepts

- A continuation in the exploration of the VB language by implementation of a real subsystem using the database structure built during the previous SQL course, with MS-SQL Express as the database behind..
- Additional practice with VB as a tool to extract data from MS-SQL, Access and CSV files
- Mission report that comments on the objectives and achievements and include recommendation about of the next steps on improving the general modeling and programmatic skills inside INE

Activities

- A handover from the recent SQL mission will be done in Copenhagen before the arrival to Maputo.
- On the Monday there will be a meeting with the counterparts on the objectives and expectations of the mission. Some work will also be done at the national accounts department, to solve any outstanding issues regarding NADABAS.
- The rest of the week: Classes / Workshops on Data modeling, VB 2005 Express and its relations to a basic SQL environment will be conduct on 4 working days (Tuesday to Friday). The workshops will be from 8.30 to 12.00. They will consist of a combination of short technical / theoretical briefings followed by hands on experience.
- A meeting towards the end of the mission with Counterparts to present and discuss the results and recommendations

Tasks to be done by INE to facilitate the mission

- Elaborate ToR for the mission
- Invite and prepare the participants of the course
- Prepare a sufficient number of computers for the training
- Prepare and supply the consultant with necessary documents and information, such as mission reports, strategies, plans etc.
- Supply good working conditions for the consultant

Consultant and Counterpart

Consultant: Søren Netterstrøm
Main counterparts: Tomas Bernardo, Anastácia Judas Honwana and more staff at DICRE/DISI.
But also Monica Magaua at DCNIG/CN.

Timing of the mission

See above.

Report

The consultants will prepare a draft report to be discussed with INE before leaving Maputo. They will submit a final draft to INE for final comments within one week of the experts have returned to work. Statistics Denmark as Lead Party will print the final version within 3+ weeks of the end of the mission. The structure of the report should be according to Danida format.

Appendix 2: A simple database application in VB.NET

This is a guide to produce a simple application in VB.NET, that will connect to a database (SQL Server or MS Access) and display data one row at a time, with the option to edit rows, insert new rows and delete rows. It is based on Visual Basic 2005 Express Edition.

Before you start, you should copy Aggregado.mdb (MS Access) to your computer.

Getting started

Start Visual Basic 2005 Express Edition and create a new windows project.

The first step is to create a data source to be used be used by the application.

For MS Access:

From the menu select **Data** and **Add new data source**

Select **Database** from the Wizards first page and press **Next**

From the next page, select Make New Connection, select Microsoft Acces as type and use Browse to locate your database (Aggregado.mdb). Press **Next.**

Answer **no** to copy the file to your project.

Next page (save connection string) just press **Next.**

On next page, select the table **Aggregado**. Press **Finish**

The wizard is now completed.

Both MS Access and SQL Server (Express)

Press the Tab for Solution explorer. This should **AggregadoDataSet.xsd.** Double click this.

You can now see how the dataset is described.

If Properties are not visible, press **F4** to view properties.

Look at the property for ID. Default vaule is <DBNull>, and this is not good. Delete it (leaving an empty string.
Do the same for all the columns.

Check that the AggregadoTable Adapter has a DeleteCommand, InsertCommand and UpdateCommand.

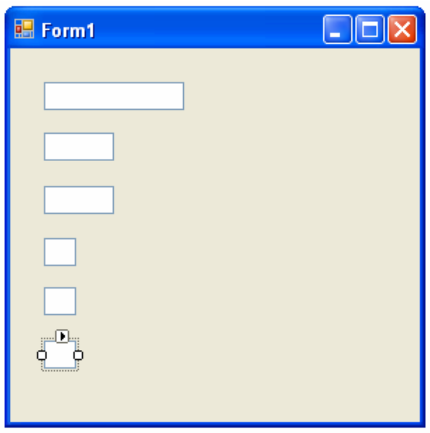
Close the window.

We have now established a data source that VB.NET will use to build the code for communication with the database.
Actually, what we have is a new namespace containing a number of new classes that we will use.

Making first part of the UserInterface

Go back to solution explorer and doubleclick on form1.VB

It should look like this



Show properties (F4) and name the boxes txtID, txtAE, txtAGREGFAM, txtF1, txtF2 and txtF3 to give them some usefull names.

Loading data

Right click the form and select Show code, to see the Code Window.

Enter code, to get this

```
Public Class Form1


    Private dataset As New
AggregadoDataSetTableAdapters.AgregadoTableAdapter
    Private datatable As New AggregadoDataSet.AgregadoDataTable
    Private row As AggregadoDataSet.AgregadoRow

    Private CurrentRow As Integer = 0

    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
        dataset.Fill(datatable)
        row = datatable.Rows(CurrentRow)
        Showdata()

    End Sub
    Private Sub Showdata()
        Me.txtID.Text = row.ID
        Me.txtAE.Text = row.AE
        Me. txtAGREGFAM.Text = row.AGREGFAM
        Me.txtF1.Text = row.F1
        Me.txtF2.Text = row.F2
        Me.txtF3.Text = row.F3
    End Sub
End Class
```

Now you start to use your new classes

Dataset is a `TableAdapter` that takes care of all communication to the Database.
Datatable is an object that contains data from the database table, as a collection of rows. This is the data we are going to use in the application.
Row is an object with data from a single row in the database.
Currentrow is used to keep track of which row is actually displayed. As we will start with the very first row, it is set to 0, because the Rows collection in the datatable (containing all the rows) is zero based.

When the form loads   dataset.fill(datatable) uses the fill method of the TableAdapter to load data into the datatable.
Then we set row to be a pointer to the first row in the table.

Showdata is made as a subroutine, as we are going to use the same code a little bit later, as we browse through the data. It does simply copy the data from the row object to the textboxes on the screen.

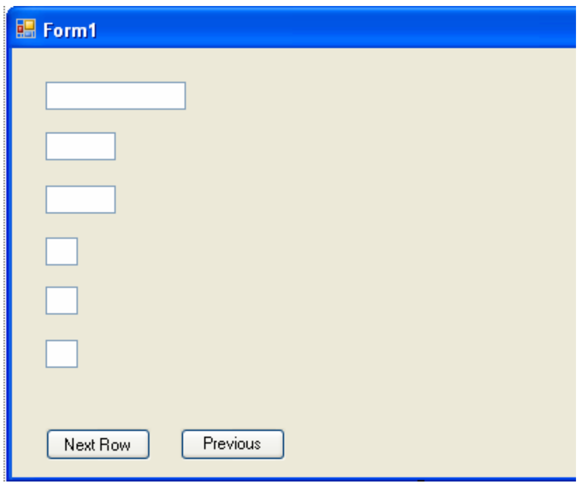
Note the Me.txtID.text. It could be   txtID.text as well. However using Me. Immideatly gives us a list of all attributes, methods and events of Me (the form), and we avoid making spelling errors for the actual names of the fields.

Test

It is now times to make the first test of the application.
As we go along, this instruction will appear frequently as it is a good idea to test each part of the application as we go, just to make sure we are on the right track.
Use the debugging options (setting breakpoint, making watches) to get more information about what is going on as needed.

When you start the application, you should se the content of the first row of the database on the screen. If you do, everything is OK.
Adding Navigation.

We will add two Pushbuttons to the screen and name them btnNextRow and bthPrevRow as this



Doubleclick Next row to go to the code and add this code:

```
    Private Sub btnNextRow_Click(ByVal sender As System.Object, ByVal
e As System.EventArgs) Handles btnNextRow.Click
        CurrentRow += 1
        row = datatable.Rows(CurrentRow)
        Showdata()
    End Sub
```

This should increase CurrentRow by 1, then change row to point to the next row and then show the data on the screen.

Similar for Previous row, go back to designer, double_click and add this code

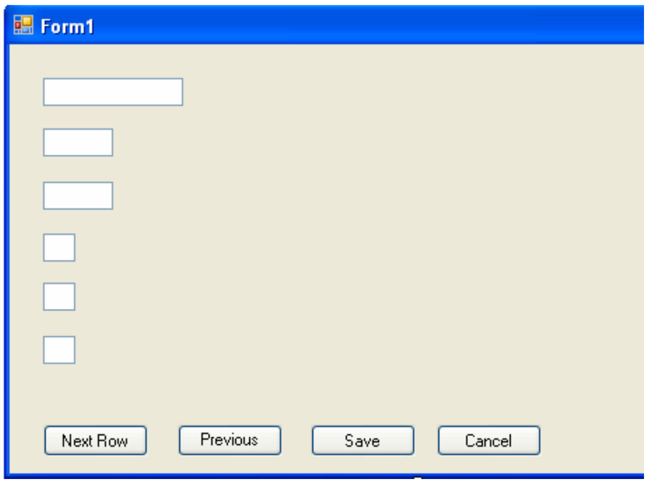
```
    Private Sub btnPrevRow_Click(ByVal sender As System.Object, ByVal
e As System.EventArgs) Handles btnPrevRow.Click
        CurrentRow -= 1
        row = DataTable.Rows(CurrentRow)
        Showdata()
    End Sub
```

Test. Click a few times on Next Row to see the data change and similar on PrevRow. If you click to many times, you are likely to get an exception, because you try to go after the last record or before the first. We will fix that problem later. The important thing here is, that you can navigate through the rows.

Handle Updates

You can change data on the screen, however, that will not in itself change the data in either **row** or **datatable** or in the database.

We need to add another set of pushbuttons, btnSave and btnCancel. With btnSave, we will save the data into the database. With btnCancel, we will restore the data using the data from row (the original data).



and as before add code

```
    Private Sub btnSave_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles btnSave.Click

        row.BeginEdit()
        row.ID = Me.txtId.Text
        row.AE = Me.txtAE.Text
        row.AGREGFAM = Me.txtAGREGFAM.Text
        row.F1 = Me.txtF1.Text
        row.F2 = Me.txtF2.Text
        row.EndEdit()
        dataset.Update(datatable)
        datatable.AcceptChanges()

    End Sub
```

This codes first copies the data from the screen back to the row (that is the datatable). It starts with **`row.BeginEdit`**`()`, to tell the row edit that we are starting to modify the row, it then copies the data and then **`row.EndEdit(),`** telling he row that we have finished to modify.
It then call the tableadapter with **`dataset.Update(datatable)`**. The tableadapter now scans the datatable for any modifications and updates the database with these changes. This may include new rows, rows to be deleted etc, as you shall se later. Finally we clean up datatable using `datatable.AcceptChanges()` that does remove all flags for changes in the datatable, since it is now synchronised with the database.

To cancel any changes use this code

```
    Private Sub btnCancel_Click(ByVal sender As System.Object, ByVal
e As System.EventArgs) Handles btnCancel.Click
        Showdata()
    End Sub
```

We simply take the original data from the datatable and put them on the screen again.

Test Modify the first record, be aware, that you keep same number of digits to avoid errors and don't modify ID (even if you could do it, but you may get duplicate key error). We handle errors later on. Press Save to modify the database, then close application and start it again, to see that the data really was changed in the database. You may also use the object browser for this. Try it out.

Test the cancel button as well.

Insert and delete records

We again add a few buttons, btnNew and btnCancel

To insert a record, we use New, that is going to give us a blank screen, where we can enter the data. We will then press Save to actually insert the data or Cancel to get rid of this and show the last record again.

Change the declarations at the top (just after currentrow) to include

```
Private InsertMode As Boolean = False
```

and insert this code

```
    Private Sub btnNew_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles btnNew.Click
        row = datatable.NewAgregadoRow
        Showdata()
        InsertMode = True
    End Sub
```

The `datatable.NewAgregadoRow` returns a row object, where all fields are filled with the default value. That's why we changed then the data source at the begging of this exercise. We then simply display this empty row. And then we use Insert mode to tell we are actually not dealing with a record from the datatable.
The row returned by `datatable.NewAgregadoRow` is not included in the datatable so far.

We now need to modify the `btnSave_Click to deal with a new row`

```
    Private Sub btnSave_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles btnSave.Click

        If Not InsertMode Then
            row.BeginEdit()
        End If

        row.ID = Me.txtId.Text
        row.AE = Me.txtAE.Text
        row.AGREGFAM = Me.txtAGRFAM.Text
        row.F1 = Me.txtF1.Text
        row.F2 = Me.txtF2.Text

        If Not InsertMode Then
            row.EndEdit()
        Else
            datatable.AddAgregadoRow(row)
            CurrentRow = datatable.Rows.Count - 1
        End If
        InsertMode = False

        dataset.Update(datatable)
        datatable.AcceptChanges()

    End Sub
```

We do not need to call `BeginEdit()`and `EndEdit()`for the new row we got in btnNew. But, after moving data to the row we need to add the row to the row collection in the datatable. This is what `datatable.AddAgregadoRow(row)` will do. It will become te last row in the collection, so we now make it the current row `CurrentRow = datatable.Rows.Count - 1` (zero based). We also tells that we are no longer in InsertMode. Then we update and accept the changes.

We also need to modify `btnCancel_Click`

```
    Private Sub btnCancel_Click(ByVal sender As System.Object, ByVal
e As System.EventArgs) Handles btnCancel.Click
        If InsertMode Then
            row = datatable.Rows(CurrentRow)
        End If
        Showdata()
    End Sub
```

If we are in Insert mode, we set row to point to the last row we displayed (that is still current row). And then show as before.

To delete a record, we simply press delete. We then need this code

```
    Private Sub btnDelete_Click(ByVal sender As System.Object, ByVal
e As System.EventArgs) Handles btnDelete.Click

        row.Delete()
        dataset.Update(datatable)
        datatable.AcceptChanges()

        If CurrentRow > datatable.Rows.Count - 1 Then
            CurrentRow = datatable.Rows.Count - 1
        End If
        row = datatable.Rows(CurrentRow)
        Showdata()

    End Sub
```

First we mark the row to be deleted, then we update the database and accept changes. No we have to display the row, that was next to the deleted row, if the deleted row was not the last. The next row actually has now moved up, so currentrow will be the number for this row. If currentrow is larger than the number of rows – 1, then it was the last row, and we display the new last row by setting currentrow.
Then set row to point to the row and show the data.

Test Try to insert some records.New, type data and Save. ID should be higher than 4 (to avoid dublicates) , the next two at most 3 chars and the last three 1 character. Close and restart to test, that the database really was updated as before. Try to delete some and test again database. Last try to insert and use cancel.

We have by now an application than has all the functionality we required from start. We can browse, edit, insert and delete records. However, the user has to take care, you can navigate outside the scope of rows, you may enter insert and then navigate, in best case getting some strange result.

We need to improve the robustness of the application, so the system will not crash do to a simple error by the user.

Making a more robust application

We are going to do this, by making sure, that we can only use the buttons where they apply. If we are showing the first record, we should not be able to navigate further back, if we show the last we should not be able to move ahead. If we have pressed new, we should only be able to use Save or Cancel to finish.
The same is true if we start to edit a record.

To do this we need first to add the following routines

```
    Private Sub EnableSave()
        Me.btnSave.Enabled = True
        Me.btnCancel.Enabled = True
        Me.btnDelete.Enabled = False
        Me.btnNew.Enabled = False
        Me.btnPrevRow.Enabled = False
        Me.btnNextRow.Enabled = False
    End Sub

    Private Sub DisableSave()
        Me.btnSave.Enabled = False
        Me.btnCancel.Enabled = False
        Me.btnDelete.Enabled = True
        Me.btnNew.Enabled = True
        EnableNavigation()
    End Sub

    Private Sub EnableNavigation()
        If CurrentRow = 0 Then
            Me.btnPrevRow.Enabled = False
        Else
            Me.btnPrevRow.Enabled = True
        End If
        If CurrentRow = datatable.Rows.Count - 1 Then
            Me.btnNextRow.Enabled = False
        Else
            Me.btnNextRow.Enabled = True
        End If
    End Sub
```

`EnableSave` enables the use of Save and Cancel, `DisableSave` disables these buttons.
If Save and Cancel are enabled, then navigation, new and delete should not be allowed.
When they are disabled, we should be able to navigate (first/last record exceptions), insert a new and delete.
`EnableNavigation` enables or disables Next and Prev reflecting whether we show the first or the last or any other record

We now start making calls to these procedures

```
    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
        dataset.Fill(datatable)
        row = datatable.Rows(CurrentRow)
        Showdata()
        DisableSave()

    End Sub
```

From the start Save and Cancel are not allowed, the others are.

```
    Private Sub btnNextRow_Click(ByVal sender As System.Object, ByVal
e As System.EventArgs) Handles btnNextRow.Click
        CurrentRow += 1
        row = datatable.Rows(CurrentRow)
        Showdata()
        DisableSave()

    End Sub

    Private Sub btnPrevRow_Click(ByVal sender As System.Object, ByVal
e As System.EventArgs) Handles btnPrevRow.Click
        CurrentRow -= 1
        row = datatable.Rows(CurrentRow)
        Showdata()
        DisableSave()

    End Sub
```

When we select a new record (Next or Prev) we need to disable save and cancel and adjust the others. When we call Showdata, then we change the fields on the screen, and as we see later, this triggers that we turn on Save.

```
    Private Sub btnSave_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles btnSave.Click

        If Not InsertMode Then
            row.BeginEdit()
        End If

        row.ID = Me.txtId.Text
        row.AE = Me.txtAE.Text
        row.AGREGFAM = Me.txtAGREGFAM.Text
        row.F1 = Me.txtF1.Text
        row.F2 = Me.txtF2.Text

        If Not InsertMode Then
            row.EndEdit()
        Else
            datatable.AddAgregadoRow(row)
            CurrentRow = datatable.Rows.Count - 1
        End If
        InsertMode = False

        dataset.Update(datatable)
        datatable.AcceptChanges(
        DisableSave()

    End Sub
```

After saving a record (and possible having inserted a new record), we adjust navigation and disable Save and Cancel.

```
    Private Sub btnCancel_Click(ByVal sender As System.Object, ByVal
e As System.EventArgs) Handles btnCancel.Click
        If InsertMode Then
            row = datatable.Rows(CurrentRow)
        End If
        Showdata()
        DisableSave()
    End Sub
```

After cancel, we are in same situation as with save.

```
    Private Sub btnNew_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles btnNew.Click
        row = datatable.NewAgregadoRow
        Showdata()
        InsertMode = True
        EnableSave()
    End Sub
```

After new, we enable save and cancel and disable all other buttons

```
    Private Sub btnDelete_Click(ByVal sender As System.Object, ByVal
e As System.EventArgs) Handles btnDelete.Click

        row.Delete()
        dataset.Update(datatable)
        datatable.AcceptChanges()

        If CurrentRow > datatable.Rows.Count - 1 Then
            CurrentRow = datatable.Rows.Count - 1
        End If
        row = datatable.Rows(CurrentRow)
        Showdata()
        DisableSave()
    End Sub
```

After delete, we make sure that navigation is OK. The new could the both last and first!. And repair Cancel and save after showdata.

```
    Private Sub txtID_TextChanged(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles txtId.TextChanged
        EnableSave()
    End Sub

    Private Sub txtAE_TextChanged(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles txtAE.TextChanged
        EnableSave()
    End Sub

    Private Sub txtAGREGFAM_TextChanged(ByVal sender As
System.Object, ByVal e As System.EventArgs) _
            Handles txtAGREGFAM.TextChanged
        EnableSave()
    End Sub

    Private Sub txtF1_TextChanged(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles txtF1.TextChanged
        EnableSave()
    End Sub

    Private Sub txtF2_TextChanged(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles txtF2.TextChanged
        EnableSave()
    End Sub

    Private Sub txtF3_TextChanged(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles txtF3.TextChanged
        EnableSave()
    End Sub
```

Finally, whenever the user start to edit, we enable save and cancel and disables all others. It does not matter if we are actually entering a new record or making multiple changes. However, when we show a new record, we actually changes these data, that's why we call disable save each time we show data.

Now the user should not be able to make wrong navigation or use of the Save and Cancel buttons.

Test

Handling data errors

Now, the user should not be able to crash the application due to making wrong navigation. However, if the user enters invalid data or makes a dublicate key, the system will still crash.

You need to handle this. Regarding wrong data, you could include intensive test for each data item, before you actually tries to load it back to the database, or even using the Validation event to catch such errors. You will however take a more simple approach that however will catch errors.

The problem arises when you try to make an update or inset a new record, that is when the user hits the Save button. New is not a problem and delete should not cause problems even.

To fix the problem, we will concentrate on save and change it in the following way.

```
    Private Sub btnSave_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles btnSave.Click

        If Not InsertMode Then
            row.BeginEdit()
        End If

        row.ID = Me.txtId.Text
        row.AE = Me.txtAE.Text
        row.AGREGFAM = Me.txtAGREGFAM.Text
        row.F1 = Me.txtF1.Text
        row.F2 = Me.txtF2.Text

        If Not InsertMode Then
            Try
                row.EndEdit()
            Catch ex As Exception
                MsgBox("Unable to edit row" + vbCrLf + ex.Message, 
vbCritical)
                row.CancelEdit()
                Exit Sub
            End Try

        Else
            Try
                datatable.AddAgregadoRow(row)
                CurrentRow = datatable.Rows.Count - 1
            Catch ex As Exception
                MsgBox("Unable to add row" + vbCrLf + ex.Message, 
vbCritical)
                Exit Sub
            End Try
        End If

        Try
            dataset.Update(datatable)
        Catch ex As Exception
            MsgBox("Unable to update row" + vbCrLf + ex.Message, 
vbCritical)
            If Not InsertMode Then
                row.RejectChanges()
            Else
                row.Delete()
            End If
            Exit sub
        End Try

        InsertMode = False
        datatable.AcceptChanges()
        DisableSave()

    End Sub
```

`row.EndEdit()`causes the datatable object to test the validity of the data, at least if keys as unique, that the length of an attribute does not exeed the maximum length etc. In case of errors, an exception is raised. We Use the `Try`, `Catch` and `End Try` to handle this. If there is an error, we inform the user with the message box, and then cancel the edit operation. Then we leave save, so we are still in insert or edit mode and the user may correct the errors and redo save or use cancel to end the edit or insert state.

`datatable.AddAgregadoRow(row)` in the same way performs validation. There is no clean up operation to perform.

`dataset.Update(datatable)` may fail for any number of reasons, so we pack it into a `Try`, `Catch` and `End Try` to handle such errors. If we cannot update, we `RejectChanges`, this will restore the state of the row as it was before the last `AcceptChanges` (last successful save) or in the case of insert, we delete the row just added (leaving it to further editing and another try).

You may from the offset think that a single `Try`, `Catch` and `End` could handle this, the problem of this would be to get things cleaned up in a proper way.

Empty database

So far, we have assumed that the database has at least one record. However, that may not be the case, we may start with a completely empty database.

To solve this problem, if we have an empty database, or delete the last row, then the system should enter Insert mode to allow us to the first record.

```
    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
        dataset.Fill(datatable)
        If datatable.Rows.Count = 0 Then
            row = datatable.NewRow
            Showdata()
            EnableSave()
        End If
        row = datatable.Rows(CurrentRow)
        Showdata()
        DisableSave()

    End Sub

    Private Sub btnDelete_Click(ByVal sender As System.Object, ByVal
e As System.EventArgs) Handles btnDelete.Click

        row.Delete()
        dataset.Update(datatable)
        datatable.AcceptChanges()

        If datatable.Rows.Count = 0 Then
            row = datatable.NewRow
            Showdata()
            InsertMode = True
            EnableSave()
        Else

            If CurrentRow > datatable.Rows.Count - 1 Then
                CurrentRow = datatable.Rows.Count - 1
            End If
            row = datatable.Rows(CurrentRow)
            Showdata()
            DisableSave()
        End If
    End Sub
```

To provide a clue to the user, we are going to change the text on the Save, if the user is in Insert Mode

```
    Private Sub EnableSave()
        If InsertMode Then
            Me.btnSave.Text = "Insert"
        Else
            Me.btnSave.Text = "Save"
        End If
        Me.btnSave.Enabled = True
        Me.btnCancel.Enabled = True
        Me.btnDelete.Enabled = False
        Me.btnNew.Enabled = False
        Me.btnPrevRow.Enabled = False
        Me.btnNextRow.Enabled = False
    End Sub
```

Test

You have now completed building a small application that allow updates on a very simple database table. A lot of things could be added, at least some labels on the user interface, the title line in the form, maybe displaying actual record number and total number of records etc. but for the purpose of this exercise that would just add noise.