CHAPTER III - CUSTOMER HOME PAGE

1. Introduction

Each customer has a personal page (see Figure 3.1), that is accessible through username and password. The personal page shows basic information (such as name, surname, type and state of the membership, etc.) and makes it possible to:

- Change Password;
- Ask for a new quotation;
- Make reservations to classes and see the list of all the active reservations;
- See special offers.

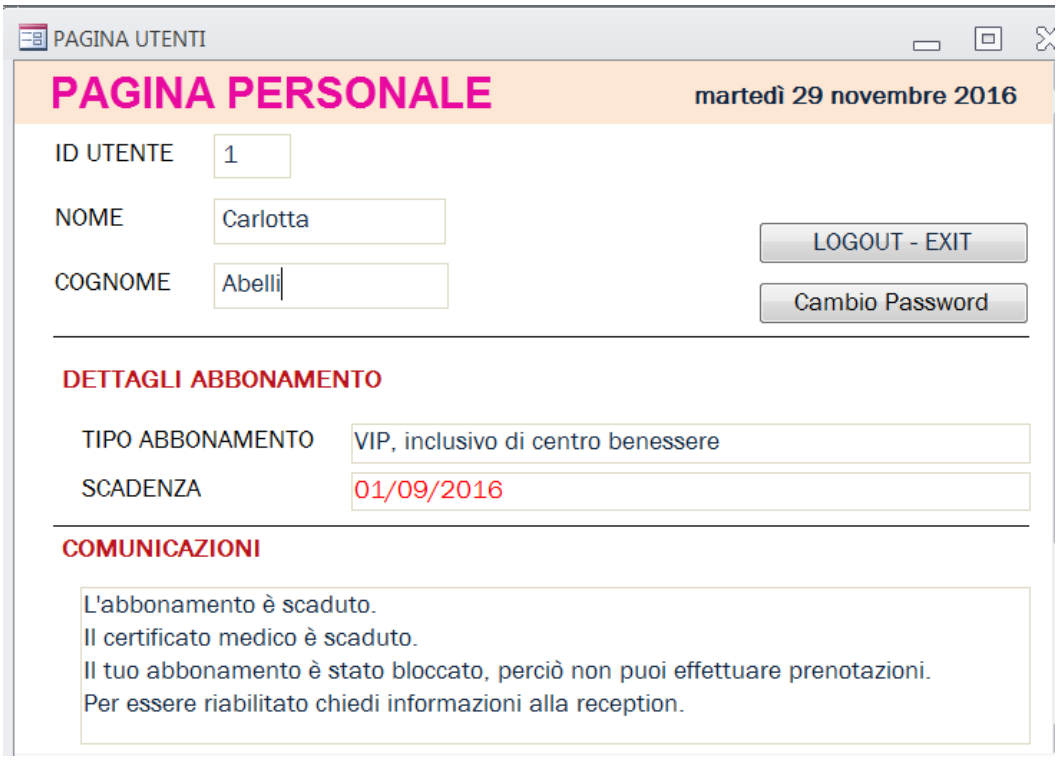

Fig. 3.1. *The customers' personal page*

Obviously this form is opened if the login was successful. In this case, the following code is executed just before opening the personal page of the customer:

```
[… Other code here …]
   WhereCon = "Username = " ' A filtering condition used in the FindFirs statement
   WhereCon = WhererCon & "'" & Me.TxtUserName & "'"
   WhereCon = WhereCon & " AND Password = '" & Me.TxtPassword & "'"
```

```
   Rs.FindFirst WhereCon      ' FindFirst looks for (and returns) the first record that complies to WhereCon
[…]
   ID = Rs![ID]  ' Saves the ID of the user
[…]
   DoCmd.OpenForm FormCustomer, , , , , , , ID
```

As it can be seen, the last statement is used to open the FormCustomer (i.e., the personal page) and, to this aim, the ID of the user is passed as the **Open Arg**.

This value is read when the form is open (i.e., on the **On Load Event** of the form) and it is used to display only the personal information of the customer that has logged in.

The code performing these actions is shown below.

```
Private Sub Form_Load()
Dim Rs As DAO.Recordset
Dim Db As Database
Dim Message As String ' A message that will be displayed on the screen
Dim MExpDate As Date ' The Expiration date of the membership
Dim DayDiff As Integer ' Number of days to the expiration
      IDUser = Me.OpenArgs ' The Open Arg is read and assigned to the IDUser global variable
  ' Now we seek the information that have to be displayed on the screen
   Set Db = CurrentDb
   Set Rs = Db.OpenRecordset("UTENTI", dbOpenSnapshot, dbReadOnly)
   Rs.FindFirst "[ID]=" & IdUser
' Data are used to fill Text Boxes (TxtXXX) located on the current Form
   Me.TxtName = Rs!Name
   Me.TxtSurname = Rs!Surname
   Me.TxtIDCustomer = Rs![ID]
' A DLookup is used to take the name of the membership card of the customer
   Message = DLookup("Name", "MEMBERSHIP", "[ID] = " & Rs![ID_Membership)
' If the user has access to the wellness area an additional message is appended to the Message String
   If Rs![Wellness Centre] = True Then Message = Message & ", wellness center is included"
   Me.TxtMembership = Messagge
   MExpDate = Expiration(Rs![Start Date], Rs![ID_Duration])  'Custom function to compute the expiring date
   Me.TxtExpiration = MExpDate
   Me.TxtExpiration.ForeColor = vbBlack ' If everything is ok, the expiration date is written in black
 ' If the expiring date is approaching (i.e., less than 15 days) ….
   If DateDiff("d", Date(), MExpDate) < 15 Then Me.TxtExpiration.ForeColor = vbRed
' Expiration date is written in red and the message is modified
   Messagge = ""
   DayDiff = DateDiff("d", Date(),MExpDate)
   If DayDiff <= 0 Then
```

```
    Message = "Please renew the membership. Validity's over."
  Else
    Message= "Membership is valid, there are " & DayDiff & " days remaining"
  End If
  If Rs!Blocked = 1 OR DayDiff < 0 Then CmdReservations.Enabled = False
  Else CmdReservations.Enabled = True
  End If
 'Check validity of the medical certificate
  DayDiff = DateDiff("d", Date(), Rs![Certificate Expiring Date])
  If DayDiff <= 0 Then
    Message = Message & vbNewLine & "Med. Certificate is not valid." 'vbnewline write in a new line
  Else
    Messagge = Message& vbNewLine & "Med. Certificate is Ok, there are " & DayDiff & "days remaining"
  End If
  If Rs!Blocked = 1 Then Message = Message & vbNewLine & "You have been blocked, go to the help desk"
   Me.TxtCommunication = Message
End Sub
```

As it can be seen, a recordset is used to get all the relevant data of the user that has logged in; these data are displayed in specific Text Boxes located on the form. In this regard, it is interesting to note that, to show the name of the membership behold by the customer the following Dlookup is used:

```
    Message = DLookup("Name", "MEMBERSHIP", "[ID] = " & Rs![ID_Membership)
```

Specifically, the name of the membership is searched in the MEMBERSHIP table; the interesting thing is that the filtering condition is obtained using the value of the ID_Membership field of the Rs recordset.

Some additional tasks are also accomplished, depending on the state of validity of both the membership and of the medical certificate. Specifically, the message written in the TxtCommunications Text Box depends on the state of validity of both the membership card and of the medical certificate. For example if the membership has expired the following message is shown: "Please renew the membership. Validity's over".

In this case also the button that allows the user to make new reservation is disabled. Also note that, to evaluate the validity of the membership a public (custom made) function is used.

Its code is shown below:

```
Public Function Expiration(D As Date, IDD As Integer) As Date
Dim Months As Integer
Dim MySQL As String
Dim Db As Database
Dim Rs As DAO.Recordset
  MySQL = "SELECT Duration FROM DURATIONS WHERE ID = " & IDD
  Set Db = CurrentDb
  Set Rs = Db.OpenRecordset(MySQL) 'In this case a DlookUp could have been used, too
```

```
   Months= Rs.Fields(0)
   Rs.Close
   Set Db = Nothing
   Set Rs = Nothing
Expiration = DateAdd("m", Months, D)
End Function
```

2. Time Schedule, Membership renewals and Offers

At the bottom of the form there are three buttons (Time Schedule, Membership Renewal and See Offers). The first one open a pivot query showing the time schedule of all courses. The second one makes it possible to ask for a quotation for a renewal of the membership. The third one shows all the offers received by the customer.

2.1 Membership Renewal - Operating on a form that is not the current one

This button opens the QUOTATIONS form that we have already described in Chapter 2. However, since in this case the user is already registered on the system, some fields of the form are automatically filled.

This is shown in the code posted below. Please note that, since modifications have to be made on another form (i.e., not on the current one), it is not possible to identify the objects (in this case text boxes) using something like Me.TxtXXX.Value. Instead, it is necessary to identify the form using the following syntax:

```
   Forms![FormName].ObjectName
```

Where Forms is the collection of all forms that are currently opened (i.e., the form must be opened before it can be modified) and Form Name is the name of the form that one wants to modify.

Also note that, in this case, the QUOTATIONS form is opened in addition modality (i.e., acFormAdd) using the customer's ID as OpenArg:

```
        DoCmd.OpenForm "QUOTATIONS", , , , acFormAdd, , IdUser
```

This makes it possible to verify if the customer has the right to get an extra discount rate due to an anticipated renewal.

```
Private Sub BtnRenewal_Click()
' This Subroutine pre-compiles the Text Box of the QUOTATIONS Form
   DoCmd.OpenForm "QUOTATIONS", , , , acFormAdd, , IdUser
   Forms![QUOTATIONS].TxtName = Nz(DLookup("Name", "CUSTOMERS", "ID = " & IdUser), " ")
   Forms![QUOTATIONS].TxtSurname = Nz(DLookup("Surname", "CUSTOMERS", "ID = " & IdUser), " ")
   Forms![QUOTATIONS].TxtEmail = Nz(DLookup("Email", "CUSTOMERS", "ID = " & IdUser), " ")
   Forms![QUOTATIONS].TxtDateOfBirth = Nz(DLookup("[Date Of Birth]", "Users", "ID = " & IdUser), " "
End Sub
```

For the sake of completeness, we also show the code executed when the QUOTATION form is open and that is triggered by the on click event of the Save button placed on that form.

```
Option Compare Database
Option Explicit
' Variables that have a visibility limited to the form
Private LoggedUser As Integer
Private Renewal As Boolean

Private Sub QuotationForm_Load()
   If Not IsNull(Me.OpenArgs) Then ' If the form is opened by a non-registered user the OpenArg is null
      LoggedUser = Me.OpenArgs
      Me.TxtCustomerID = Me.OpenArgs
      Renewal= True
   End If
End Sub

Private Sub CmdPreventivi_Click()
' The other part of the code is shown in Section 2.2. of Chapter 2
[…]
  If Renewal Then ' Advance is true if the user has already a membership card
      AdDis = 0
      MySQL = "SELECT DURATIONS.Duration, CUSTOMERS.[Starting Date] FROM "
      MySQL = MySQL & " DURATIONS INNER JOIN CUSTOMERS ON DURATIONS.ID = CUSTOMERS.ID_Duration"
      MySQL = MySQL & " WHERE CUSTOMERS.ID = " & LoggedUser
      Set Rs = Db.OpenRecordset(MySQL)
      ' We compute the expiration date taking the sum of the starting date and of the duration
      Expiration = DateAdd("m", Nz(Rs.Fields(0), 0), Nz(Rs.Fields(1), Date))
      Advance = DateDiff("m", Date(), Expiration) ' Months to the expiration date i.e., renewal advance
      Rs.Close
      ' We Compute the extra discount rate
      Set Rs = Db.OpenRecordset("SELECT Condition, Discount FROM DISCOUNTS")
      Rs.Move (3) ' Records 4 to 6 define different discount rates, depending on the length of the advance
      If Advance >= Rs.Fields("Condition") Then
         AdDis = Rs.Fields("Discount")
         Rs.MoveNext
         If Advance >= Rs.Fields("Condition") Then
            ScAnt = Rs.Fields("Discount")
            Rs.MoveNext
            If Advance >= Rs.Fields("Condition") Then
               ScAnt = Rs.Fields("Discount")
               Rs.MoveNext
            End If
         End If
```

```
      End If
      Discount = Discount + AdDis
   End If
[…]
End Sub
```

2.2 Show Offers - Opening a Form with a where condition used to filter data

This button opens a form showing all the offers made to the customer. The form is dynamically linked to the OFFERS table. So, using a simple statement such as Docmd.OpenForm "OFFERS" would be wrong. In this case, in fact, the user would get read and write access to all the offers generated for all the customers of the Sport Centre. To avoid this behavior, the form is opened in "read-only" mode and a filter is used to limit the visibility to the records containing the offers for the user that has logged in. Specifically, the following statement is used:

```
   DoCmd.OpenForm "OFFERS", , , WhereCond, acFormReadOnly
```

Also note that **WhereCond** is a string containing a filtering condition and **acFormReadOnly** specify that the access grant to the user is of "read only" type.

```
Private Sub CmdOffers_Click()
Dim WhereCond As String
  ' Only the offers (of the customer logged in) that are still valid are show
  WhereCond = "UserID = " & LoggedUser & " AND [Expiration Date] >= "
  WhereCond = WhereCond & Format(Date, "\#mm\/dd\/yyyy\#") '#" & Date & "#"
  DoCmd.OpenForm "OFFERS", , , WhereCond, acFormReadOnly
End Sub

Private Sub Form_Load()
On Error GoTo Err
DoCmd.GoToRecord , , acLast   ' The last records is shown
Err:
   If Err.Number <> 0 Then  ' In case of error code execution is not abruptly stopped, but a message is displayed
MsgBox ("You do not have any valid offer")
      DoCmd.Close acForm, "OFFERS"
   End If
End Sub
```

Note that the **GotoRecord** methods of the **DoCmd** Object is used to show, the last one of the valid offers. This is made with the following instruction:

```
   DoCmd.GoToRecord , , acLast
```

where: acLast specifies that the last record must be shown, at first.

Clearly in case of multiple offers (that are still valid) the user is free to navigate among them.

It is also important to note that the **On Error Goto** condition is essential because, due to the filtering condition, a customer may not have any valid offer. In this case the GoToRecord method would rise an error (since no records have been found). So, to avoid a crash of the program the code is diverted to the part written below the Err string; this part is used to "manage the error" displaying a "gentle" error message on the screen. We also recall that the part of the code placed below the Err string is always executed, even if an error has not occurred. So, not to display the error message, an If … Then condition, based on the Err.Number value is used. This is shown below:

```
[…]
 Err:
    If Err.Number <> 0 Then  ' In case of error code execution is not abruptly stopped, but a message is displayed
          MsgBox ("You do not have any valid offer")
          DoCmd.Close acForm, "OFFERS"
    End If
 […]
```

The code works because, anytime an error occurs, the Number property of the Err object takes a value equal to the numeric code that codifies the occurred error. In other words, in case of error, the Err.Number is positive, and it is zero otherwise.

3. Reservation list (Dettagli Corsi)

At the bottom of the personal page there are some buttons (not shown in Figure 3.1). One of this, namely Reservation list, opens a form showing the list of all the reservations made by the logged customer. This form, shown in Figure 3.2 is one of the most complex of the information system.

Fig. 3.2 *Reservations' List*

As it can be seen, the form shows both the list of the "*active reservations*" and the list of the "*pending reservations*". The last one contains the classes that, at the time of the reservation, were already full, and therefore the customer was added to a waiting list.

The user can select a class to delete the reservation or to leave the waiting list. If the user selects an active reservation, the selected class is highlighted in red and the "leave queue" button in disabled. Conversely, if the user selects a pending reservation, the selected class is highlighted in blue and the "Cancel Reservation" button is disabled.

Obviously, using the "New reservation" button, the user can see the list of all the classes (scheduled within the next seven days) that he or she can join, to make a new reservation.

3.1 The Sub Forms - Using conditional formatting and filtering condition

Before proceeding further on, it is important to note that the form is based on two sub forms, both of multi items type called, respectively, Reserved Classes (Corsi Prenotati) and Pending Classes (Corsi in Lista).

In the next part of this Sub-Section we will consider only the Reserved Classes form, since the other one has exactly the same structure and VBA code.

Specifically, the Reserved Classes form is linked to the following query:

```
SELECT RESERVATIONS.ID AS Reservation, RESERVATIONS.CustomerID, COURSES.Name, _
                RESERVATIONS.[Reservation Date], RESERVATIONS.[Class Date], RESERVATIONS.[Class Time]
FROM COURSES INNER JOIN RESERVATIONS ON COURSES.ID = RESERVATIONS.CourseID
WHERE RESERVATIONS.[Class Date] >=Date() AND RESERVATIONS.[Reservation State] = 1
ORDER BY RESERVATIONS.[Class Date]
```

As it can be seen, the query returns all the active reservation of all the classes that are scheduled from now on. Also note that, in the query, the user ID it is not used as a filtering condition. This filter will be executed at run time, when the Reserved Class form will be opened (i.e., when the load event takes place).

In the preceding sections we have seen that a filtering condition can be defined directly in the OpenForm method of the Docmd object. However, in this case, for the sake of completeness we will use another approach and we will apply a filter to the form using the **Filter** and the **FilterOn** methods of a form object. This is shown below:

```
Private Sub Form_Load()
Dim Flt As String
Dim LoggedUSer As Integer
LoggedUser = Me.Openarg
Flt = "[ID UTENTE] = " & LoggedUser ' A very simple filtering condition
Me.Filter = Flt ' We assign the filter
Me.FilterOn = True ' We activate the filter
```

```
Me.Requery ' We refresh/update the database, so that the effect of the filter is shown on the form
End Sub
```

the last instruction **Me.Requery** is used to update the form, so that only the filtered records will be displayed

As we have anticipated above, we also want to highlight, in red, the record (i.e., the reservation) selected by the user. To this aim we will take advantage of a **Conditional Formatting Rule**.

More precisely, we must create a conditional formatting rule similar to the one that is graphically shown in Figure 3.3; briefly this rule says that if a "Reservation" field has the same value of the ID of the field selected by the user, then it must be colored in red.

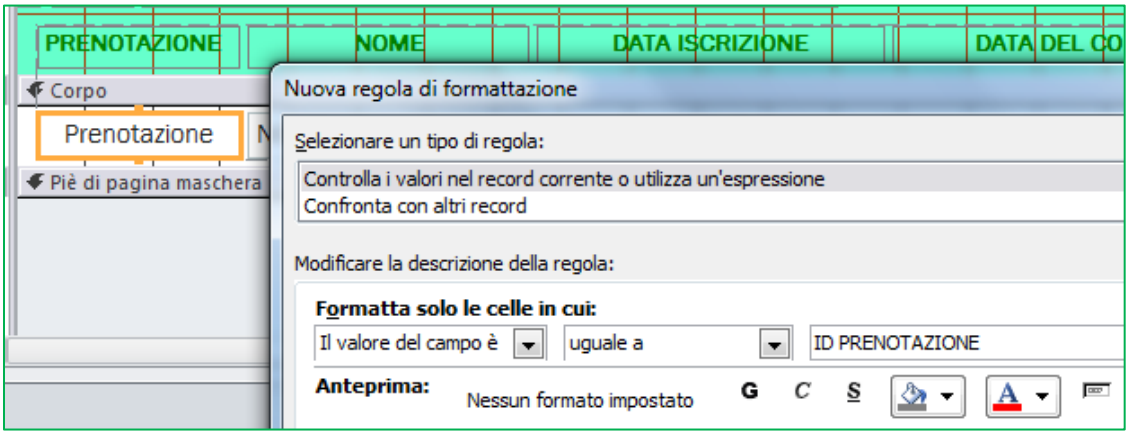

Fig. 3.3 *Conditional Formatting*

Is needless to say that the condition of Figure 3.3 is static; conversely, we need a dynamic rule, since the comparison value must be changed any time the user modifies his or her selection. To this aim we need to add some lines of code that must be triggered by the **OnCurrent** Event, an event that occurs any time something changes on the form. The full code is shown below.

```
Private Sub Form_Current()
Dim Frm As FormatCondition
Dim ID As Integer
On Error Resume Next ' If there are no booked classes the condition ID = Reservation would rise an error
ID = Reservation ' The Value of the Reservation field is assigned to the ID variable
' We delete ALL the previous conditional formatting rules and we add a new one
    If Me.Reservation.FormatConditions.Count > 0 Then Me.Reservation.FormatConditions(0).Delete
    Set Frm = Me.Reservation.FormatConditions.Add(acFieldValue, acEqual, ID)
' Now a conditional formatting rules, based on the ID value exists.
' We only need to say what must happen anytime the rule is triggered
      If Me.Reservation.FormatConditions.Count > 0 Then
          Me. Reservation.FormatConditions(0).ForeColor = vbRed  ' Text in red
          Me.Prenotazione.FormatConditions(0).Enabled = True ' Rule is activated
       End If
End Sub
```

The functioning of the Subroutine is explained below:

- Since the form is a multi-item form, Reservation takes the value contained in the Reservation Text box of the record selected by the customer;
- This value is assigned to the ID integer variable;
- In case the field Reservation already had a conditional rule, we delete it with the following statement:

```
If Me.Reservation.FormatConditions.Count > 0 Then Me.Reservation.FormatConditions(0).Delete
```

Where:

Count returns the number of rules of the **FormatConditions** collections

FormatConditions(0).Delete erases the first and, in this case, the only existing rule

- We assign to the FormatCondition property (of the Reservation field of the Form) a new format condition. This is made with the following statement:

```
Set Frm = Me.Reservation.FormatConditions.Add(acFieldValue, acEqual, ID)
```

Where:

Add(acFiedlValue, acEqual, ID) is used to define the rule

the rule is based on the value of the field (i.e., acFieldValue)

this value must be equal to ID (i.e., acEqual, ID)

- We define the behavior of the rule and we activate it, with the following statements:

```
Me. Reservation.FormatConditions(0).ForeColor = vbRed  ' Text in red
Me.Prenotazione.FormatConditions(0).Enabled = True ' Rule is activated
```

3.2 The Main Form - Using the recordset cloning technique

As we mentioned above, both Reserved Classes and Pending Classes are included as sub forms (respectively, Sub1 and Sub2) of the main form Reservations List.

This is shown, in design view, in figure 3.4.

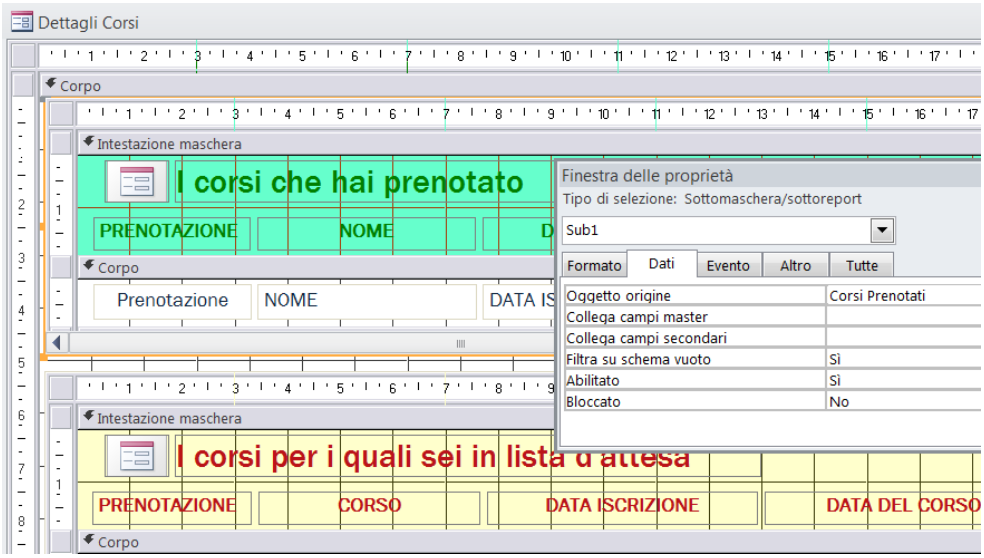

Fig. 3.4 *The main form in design view*

As it can be seen (the properties window is relative to the first sub form), the first sub form is linked to the Reserved Classed form and, similarly, the second sub form is linked to the Pending Classes form.

This is just the standard and easiest part of this form. The real novelty is the fact that, to manipulate both sub forms, two recordsets RsSub1 and RsSub2 will be used to clone[1], the Reserved Classes and the Pending Classes forms, respectively.

For the sake of clarity, we recall that, anytime a form is linked with a table or with a saved query (either in a mono directional (Snapshot) or bidirectional (Dynaset) way), behind the scenes Access creates a hidden recordset used to manage the connection between the form and the table or the query to which the form is linked to. The important thing is that it is possible to assign, using the recordset.clone method, this hidden recordset to an explicit one; this procedure makes it possible to perform checks and dynamic changes.

[1] *See Chapter II for some insights on the cloning method*

The following code, triggered by the on load event of the form, explains how this technique can be effectively used.

```
Option Compare Database
Private RsSub1 As DAO.Recordset, RsSub2 As DAO.Recordset ' Recordsets with form visibility
Private Sub Form_Load()
'Two clones are created.
   Set Rs1 = Forms![Reservations List]![Sub1].Form.RecordsetClone  ' The path to the sub form
   Set Rs2 = Forms![Reservations List]![Sub2].Form.RecordsetClone ' Both "considered" as Forms
 ' Both the buttons to cancel a reservation are disabled; to be re-enabled the user must select a class
   Me.CmdCancReservation.Enabled = False
   Me.CmdCancQueue.Enabled = False
' Conditional formatting is enabled in both sub-forms
   Forms![Reservations List]![Sub1].Form!Reservation.FormatConditions(0).Enabled = True
   Forms![Reservations List]![Sub2].Form!Reservation.FormatConditions(0).Enabled = True
End Sub
```

Note that:

- The recordsets are declared as private variables with a visibility limited to this form. This is because they will be used in many parts of the code. In technical language we say that these objects have a visibility limited to the form in which they reside.
- To create the clones we do not need to use a Data Base variable (i.e., Set Db = CurrentDb), because the original (hidden) recordsets have been automatically created (by Access at the opening of the form) and so we can just point to them with the following instruction_

  ```
  Set Rs1 = Forms![Reservations List]![Sub1].Form.RecordsetClone
  ```

  It is worth making a brief comment about the expression: Forms![Reservations List]![Sub1]

  Here:

  - Forms is the collection of all the opened forms;
  - Reservation List is the name of the form (of the collection) to which we want to refer (i.e., the current form);
  - [Sub1] is the sub form owned (as denoted by the exclamation mark !) by the main form.
- So, up to here we have identified the object and the sub-form that we want to use. Actually, we could have simplified the code using the expression Me.Sub1 rather than Forms![Reservations List].
- Now that we have identified the object that we are interested in, we must specify how we want to usethis object. This is made with the second part of the instruction:

  ```
  .Form.RecordsetClone
  ```

This part it's quite odd. Why do we need to re-write the term Form? The reason is simple; we have to specify the type of object that we are using:

- Sub1 is a form and so, to get access to all its properties and methods, we need to explicit that it is a form;
- Only after doing that, we can, finally, use the RecorsetClone method to set our "explicit" recordset RsSub1.

Now that we have a clone, we can take advantage of all the features of the Reserved Class form. This is shown in the following code, triggered by the On Enter event, that is executed anytime the users place the mouse's cursor inside a sub-form.

```
Private Sub Sub1_Enter()
' The On Enter event is activated when the user places the mouse's cursor inside a sub form
  If RsSub1.RecordCount > 0 Then ' We use the clone to see if there are active reservations
     Me. CmdCancReservation.Enabled = True
     Me.CmdCancQueue.Enabled = False
     Forms![Reservations List]![Sub1].Form!Reservation.FormatConditions(0).Enabled = True
     Forms![Reservations List]![Sub2].Form!Reservation.FormatConditions(0).Enabled = False
  End If
End Sub
Private Sub Sub2_Enter()
 ' Exactly as before, with Sub1 and Sub2 reversed
  If RsSub2.RecordCount > 0 Then
     […]
  End If
End Sub
```

As it can be seen the recordset clone RsSub1 is used to count the number of active reservations. If there are some active reservations and the user click on one of them, the button that allows deleting a reservation is enabled, whereas the one used to leave the queue is disabled. Similarly, the conditional formatting is activated only for the Sub1 Sub Form and it is de-activated for the Sub2 Sub Form. In this way, the selected active class becomes red and, conversely, all the pending classes are in black.

When a class is selected the cancel button is enabled and the reservation can be cancelled. For the sake of brevity, only the code relative to the CmdCancReservation button is shown; the other one is almost the same.

```
Private Sub CmdCancReservation_Click()
   Call Delete
   If RsSub1.RecordCount = 0 Then Me.CmdCancReservation.Enabled = False
End Sub

Public Sub Delete()
Dim Res As VbMsgBoxResult
Dim ResID As Integer
Dim MySQL As String
   On Error Resume Next
   ' We take the ID of the reservation that has to be deleted
   ResID = Forms![Reservations List]![Sub1].Form![Reservation]
   If Me.ActiveControl.Name = "CmdCancQueue" Then
         ResID = Forms![Reservations List]![Sub2].Form![Reservation]
  End If
 ' A Confirmation message is displayed on the screen
   Res = MsgBox("You selected Reservation N° " & ResID & vbNewLine & "Do you confirm erasing?", _
                                                                    _vbYesNo, "Erase")
' If the user clicks on yes a delete query is executed
  If Res = vbYes Then
      MySQL = "DELETE * FROM RESERVATION WHERE ID = " & ResID
      DoCmd.SetWarnings False
      DoCmd.RunSQL MySQL
      DoCmd.SetWarnings True
      Forms![Reservations List]![Sub1].Form.Requery ' Records displayed on the form are updated
      Forms![Reservations List]![Sub2].Form.Requery
   End If
End Sub
```

As it can be seen, the CmdCancReservation_Click is based on the Delete subroutine that works for the CmdCancQue_Click, too. After the reservation has been canceled, the cloned recordset is used to count the remaining reservations. If there are no more reservations, the cancel button is disabled.

Concerning the Delete subroutine, we note that, at first ResID takes the value of the selected reservation:

```
   ResID = Forms![Reservations List]![Sub1].Form![Reservation]
```

However, if the user had selected a pending reservation this assignment would raise an error. This is avoided thanks to the use of the On Error Resume Next code that pushes the execution to the next line. Here an If…Then condition is used to see if the user clicked on the Leave Queue button and, if so, ResID takes the value of the selected pending reservation. Please note that, in this case the following statement is used:

```
    If Me.ActiveControl.Name = "CmdCancQuee" Then
```

In other words we are checking that the user has clicked on the Leave Queue button.

Anyhow, after the user clicks one of the two button a confirmation button is displayed on the screen. This is made with the following assignment.

```
   Res = MsgBox("You … " & ResID & vbNewLine & "Do you …?",   vbYesNo, "Erase")
```

This assignment could seem odd,: why should assign a Message Box to a variable? However, there it is nothing strange in it: Res if a variable of type VbMsgBoxResult, i.e., a variable that stores the action performed by the user on the message box. In this case, since we have specified with the second input parameter (i.e., vbYesNo) that the message box must have both a Yes and a No buttons, Res will get a vbYes value if the user clicks on the yes button and, it will get a vbNo value, otherwise

Lastly a delete query is executed using the RunSQL method of the Docmd object.

We conclude this section by noting that, once the form is closed, both recordsets are closed and killed to free up memory. This is shown below.

```
Private Sub Form_Close()
On Error Resume Next
   RsSub1.Close
   RsSub2.Close
   Set RsSub1 = Nothing
   Set RsSub2 = Nothing
End Sub
```

4. Making a new Reservation - Available Classes (Corsi Prenotabili)

Clicking on New Reservations the form Bookable Classes is displayed; as shown by Figure 3.5 this form shows all the classes (scheduled within the next seven days) that can be booked by a customer.

Fig. 3.5 *Bookable Classes*

This form is linked to the saved query Time-Tables With Dates, that has been already described in Section 4 of Chapter I. Briefly we recall that, this query, collects data from COURSES and TIME TABLES and, using a set of user defined VBA functions (highlighted in red) also computes the date of a class, the number of vacancies and the number of people that made a reservation.

```
SELECT COURSES.*, [TIME TABLE].*, _
        ClDate([TIME TABLE].Day) AS Date,_
        FrPlaces([TIME TABLE].ID_Course, [TIME TABLE].Day, [TIME TABLE].[Start Time]) AS [Free Places],_
        TotReg([TIME TABLE].ID_Course, [TIME TABLE].Day, [TIME TABLE].[Start Time]) AS [People In],_
        InQueue([TIME TABLE].ID_Course, [TIME TABLE].Day, [TIME TABLE].[Start Time]) AS [People Waiting],_
FROM COURSES INNER JOIN [TIME TABLE] ON COURSES.ID = [TIME TABLE].ID_Course
```

4.1 Filtering and Sorting Data on Form Open

Since the query does not contain a WHERE condition, it returns all the classes that will take place within the next seven days[2]. However, the logged customer may not be allowed to join all classes, as classes that can be attended depend on the type of his membership card. Thus, we will execute this filter at the opening of the form:

[2] The fact that classes are limited to the next seven days is due to the fact that TIME TABLE contains a weekly schedule

```
Dim FilterT As String ' The Timing filtering condition, hold in a variable with local visibility
Private Sub Form_Load()
Dim Db As Database
Dim Rs As DAO.Recordset
Dim Ts As Date, Te As Date
Dim Earth As Boolean, Aqua As Boolean, Swim As Boolean
Dim FilterCT As String, Sorting As String, MySQL As String ' The filter (course and time), SQL and sorting criteria
Dim RN As Integer
Dim NewValList As String ' The list of times, course types etc. used to populate the filtering combo boxes
On Error Resume Next
    ' We need to understand which classes can be joined by a customer
    Set Db = CurrentDb
  MySQL = "SELECT CUSTOMERS.ID, [TIME WINDOWS].[Start Time], [TIME WINDOWS].[Ending Time], _
             MEMBERSHIPS.[Earth Course], MEMBERSHIP.[Swimming Courses], MEMBERSHIP.[Water Courses]"
  MySQL = MySQL & " FROM MEMBERSHIP INNER JOIN ([TIME WINDOWS] INNER JOIN CUSTOMERS_
               ON [TIME WINDOWS].ID = CUSTOMERS.[ID Time Window]) _
               ON   MEMBERSHIPS.ID = CUSTOMERS.[MembershipID]"
  MySQL = MySQL & " WHERE CUSTOMER.ID = " & LoggedUser ' This is a global variable
  Set Rs = Db.OpenRecordset(MySQL)
  ' We read the values
    Ts = Nz(Rs.Fields("[Start Time]"), "07:00:00")
    Te = Nz(Rs.Fields("[Ending Time]"), "23:00:00")
    Earth = Nz(Rs.Fields("[Earth Courses]"), False)
    Aqua = Nz(Rs.Fields("[Water Courses]"), False)
    Swim = Nz(Rs.Fields("[Swimming Courses]"), False)
  Rs.Close
  Set Db = Nothing
  Set Rs = Nothing
  ' We create two filters: one for the accessing time, one for courses' typology
  FilterT = "[Start Time] >= #" & Ts & "# AND [Start Time] < #" & Te & "#"
  If Acqua Then
    FilterCT = FilterT & " AND [Category] = " & "'" & "Water Courses" & "'"
    NewValList = NewValList & Chr(34) & "Aqua" & Chr(34) & ";"   ' Chr(34) = " "
   End If
  If Swim Then
    FilterCT = FilterT & " AND [Category] = " & "'" & "Swim Courses" & "'"
    NewValList = NewValList & Chr(34) & "Swim" & Chr(34) & ";" ' Courses to be included in the Combobox
  End If
  If Earth Then
    FilterCT = FilterT & " AND [Category] = " & "'" & "Earth Courses" & "'"
    NewValList = NewValList & Chr(34) & "Earth" & Chr(34) & ";"
  End If
  Me!CmbType.RowSourceType = "Value List" ' Fields of the Combo Box are taken from a list of values
```

```
  Me!CmbType.RowSource = NewValList ' We assign the list containing the values
  If Me. CmbType.ListCount >= 1 Then
     Me. CmbType.Value = Me. CmbType.ItemData(Me. CmbType.ListCount - 1) ' The last item is shown
  Me.CmbDay.Value = Me.CmbDay.ItemData(0) ' We show the first value of the list “Every day”
  Me.Filter = FilterFT   ' The filter is assigned
  Me.FilterOn = True  ' The filter is activated
  ' We also create a sorting condition
  Sorting = "[Class Date] ASC,[Class Time] ASC, [Name] ASC, [Free Places] DESC"
  Me.OrderBy = Sorting ' The sorting condition is assigned
  Me.OrderByOn = True ' The sorting is activated
  Me.Requery  ' The form is updated
End Sub
```

It is interesting to note that, at first, a filtering condition that operates only on the access period is generated. This filter is assigned to the variable FilterT, which has a local visibity. This is because this filtering condition is fixed (i.e., it depends on the type of membership owned by the logged user) and cannot be changed by the user at run time. Next a filtering condition operating both on “time” and on “courses” (i.e., FilterTC) is created, by appending to FilterT an additional condition concerning the type of course that can be joined by the customer. This is made with the following three If … Then conditions:

```
[…]
If Acqua Then
     FilterCT = FilterT & " AND [Category] = " & "'" & "Water Courses" & "'"
     NewValList = NewValList & Chr(34) & "Aqua" & Chr(34) & ";"   ' Chr(34) = “ ”
   End If
  If Swim Then
     FilterCT = FilterT & " AND [Category] = " & "'" & "Swim Courses" & "'"
     NewValList = NewValList & Chr(34) & "Swim" & Chr(34) & ";"
  End If
  If Earth Then
     FilterCT = FilterT & " AND [Category] = " & "'" & "Earth Courses" & "'"
     NewValList = NewValList & Chr(34) & "Earth" & Chr(34) & ";"
  End If
[…]
```

Let us suppose that a customer has only the right to join Aqua courses at lunch time. In this case only the first If condition will be executed and the condition will be something like:

```
[Start Time] >= #12:30# AND [Start Time] <= #14:30# AND Category = 'Water Courses'
```

Now, what if the customer can join all types of courses at lunch time?

In this case all the if conditions are true, but, since every time the FilterCT is overwritten, at the end, the following filtering condition (relative to the last If condition) will be obtained:

```
[Start Time] >= #12:30# AND [Start Time] <= #14:30# AND Category = 'Earth Courses'
```

So, at first the customer can see only earth courses. What about the other ones? At the bottom of the form there are two commands that make it possible to filter data in terms of courses' category and day. So, if a customer has access to all type of course, at first he will see only the earth courses, but he can easily visualize all the other ones using the Combo Box CmbType to modify the filtering condition.

Clearly, also the values contained in this Combo Box depend on the type of membership owned by the customer. In order to dynamically update the list, the **RowSourceType and the RowSource properties** of the Combo Box are used. The first indicate where data are collected, for instance typing Value List, values are taken form a list of string separated by semicolon (;). The second one defines the List of values to be used.

That is:

```
Me!CmbType.RowSourceType = "Value List" ' Fields of the Combo Box are taken from a list of values
Me!CmbType.RowSource = NewValList ' We assign the list containing the values
```

In order to property create the Value List, we assign a value (separated by a semicolon) to the string variable NewValList inside each If…Then condition. For instance, if a customer can access all type of courses, and so all the If conditions are true, at the end, NewValList will be something like: {Earth; Aqua; Swim}.

Concerning the Combo Box with the day, since the list of the day does not depend on the membership owned by the customer, the following value list {every day; Monday; …; Sunday} has been assigned operating, directly, on the property windows of the CmbDays that can be accessed in Design View.

Lastly, we note that, at the end of the code also a sorting condition is created:

```
Sorting = "[Class Date] ASC,[Class Time] ASC, [Name] ASC, [Free Places] DESC"
Me.OrderBy = Sorting
Me.OrderByOn = True
```

In this way classes are ordered in terms of date then of starting time then in alphabetical order and, lastly, in terms of free places.

4.2 Filtering and Sorting Data when the form is already opened

Clicking on the Filter button the user can filter data using, as filtering criteria, the values shown in the "Course Type" and in the "Days" Combo Boxes. The code, very similar to the previous one, is reported below, without any additional comments.

```
Private Sub CmdFilter_Click()
Dim Filt As String, Sort As String
  Select Case Me.CmbType.Value
    Case "Earth"
```

```
        Filt = FilterT & " AND [Category] = " & "'" & "Earth Courses" & "'"
    Case "Aqua"
        Filt = FilterT & " AND [Category] = " & "'" & "Water Courses" & "'"
    Case "Swim"
        Filt = FilterT & " AND [Category] = " & "'" & "Swim Courses" & "'"
  End Select
  If Me.CmbDay.Value <> "Any Day" Then
        Filt = Filt & " AND [Class Day] = " & "'" & Me.CmbDay.Value & "'"
  Me.Filter = Filt
  Me.FilterOn = True
  Sort = "[Class Date] ASC,[Class Time] ASC, [Name] ASC, [Free Places] DESC"
  Me.OrderBy = Sort ' The sorting condition is assigned
  Me.OrderByOn = True ' The sorting is activated
  Me.Requery  ' The form is updated
End Sub
```

4.3 Make a new reservation

The form allows the user to make a new reservation; to this aim it is sufficient to click on the “Book” button placed at the immediate right of the selected class. The code, which has nothing new, is shown below.

```
Private Sub CmdBook_Click()
Dim ID As Integer, IDBk As Integer, BkType As Integer
Dim Res As VbMsgBoxResult
Dim Condition As String, Messagge As String
Dim SqlAppend As String
  BkType = 1 ' Standard, 0 stands for waiting list
  ID = Nz(DLookup("ID", "COURSES", "Name = " & "'" & Me.Name & "'")) ' ID of the selected Course
  ' We check that the class has not been already booked by the same customer
  Condition= "CourseID = " & ID
  Condition = Condition & " AND CustomerId = " & LoggedUser
  Condition = Condition & " AND [Class Date] = " & Format(Me.Data, "\#mm\/dd\/yyyy\#")
 IDBk = Nz(DLookup("ID", "RESERVATION", Condition))    ' Search, if it exist, the ID of the prenotatin
    If IDPBk <> 0 Then ' If <> 0 it is null i.e., not found
        MsgBox "You already signed in for this course. Reservation n° " & IDBk, vbInformation
        Exit Sub
  End If
' Otherwise
  Messagge = "Do you want to sign in"
  If Me.FreePlaces = 0 Then
        Messagge= Messagge & "but in waiting list"
        BkType = 0
```

```
  End If
  Messagge = Messagge& "to " & Me.Nam & " held on " & Me.Date & "?"
  Res = MsgBox(Messagge, vbYesNo, "Booking")
  If Res = vbYes Then ' In this case an append query is executed to add a record to the RESERVATIONS Table
  ' At first we define the fields that must be added
        SqlAppend = "INSERT INTO RESERVATIONS (UserID, CourseID, [Insription Date], _
                    [Inscription State], [Class Date], [Class Time])"
    ' Next the values of the fields are defined
        SqlAppend = SqlAppend & " VALUES (" & LoggedUser & ", " & ID & ", " & _
                    Format(Date, "\#mm\/dd\/yyyy\#")& ", "
         SqlAppend = SqlAppend & BkType & ", "& Format(Me.Date, "\#mm\/dd\/yyyy\#") & ", #" _
                    & Me.Time & "#)"
     DoCmd.SetWarnings False
     DoCmd.RunSQL (SqlAppend)
     DoCmd.SetWarnings True
     On Error Resume Next
     Me.Requery       ' We update everything
     Forms![Reservation List].Requery
     Forms![ Reservation List]![Sub1].Form.Requery
     Forms![ Reservation List]![Sub2].Form.Requery
  End If
End Sub
```

4.4 Change Password

The form is also equipped with a button that opens a form that allows the user to change its password. The code is very similar to the one used for the log in. The only noticeable difference is due to the fact that, this time, the information concerning Username, Password and user type are not read from text boxes but are passed as multiple openarg parameters. This is shown in the code below.

```
Private Sub CmdGoChangePW_Click()
 ' On click, we open the form that allows the user to change his or her password
  Dim OpArg As String
  OpArg = LoggedUser & "|" & "Customers"
  DoCmd.OpenForm "CHANGE PASSWORD", , , , acFormAdd, , OpArg
End Sub

Private Sub Form_Load()
' On loading, of the Change Password form, we perform some actions
Dim LInput() As String
Me.TxtUser.SetFocus
Me.CmdChange.Enabled = False ' The change password button is disabled
     LInput = Split(Me.OpenArgs, "|") ' Split is used to split the string in two parts, i.e., ID and Customer
     Identification = LInput(0) ' Variable with visibility limited to the form
```

```
    Table = LInput(1) ' Variable with visibility limited to the form
End Sub

Private Sub CmdChangeOwd_Click()
Dim Rs As DAO.Recordset
Dim Query As String
On Error GoTo ErrorLine
  Query = "SELECT Password FROM " & Table & " WHERE [ID] = " & Identification
  Set Rs = CurrentDb.OpenRecordset(Query)
  If Not Rs Is Nothing Then
     Rs.Edit
     ' A procedure to verify the unicity of the password and to check also its robustness should be advisable
     Rs.Fields(0) = Me.TxtNewPw.Value
     Rs.Update
     Rs.Close
     Set Rs = Nothing
     Me.txtPassword.Value = Me.TxtNewPw.Value
     Me.TxtNewPw.Value = ""
     Me.TxtConfPw.Value = ""
     Me.CmdConf.Enabled = False
     MsgBox ("Password updated")
  Else
    MsgBox "Warning! Password non updated"
  End If
ErrorLine:
  If Err.Number <> 0 Then
      MsgBox "Warning ! The following error was generated:"  Err.Number & " " & Err.Description, vbCritical, ""
  End If
End Sub
```