

CHAPTER V - ADMINISTRATORS' HOME PAGE

1. Introduction - Creating a navigation Form

When an administrator logs in, he or she is diverted to a navigation form (see Figure 5.1) shared by all the administrators. The navigation form gives access to four different pages:

- *Dashboard (Intro Amministratori)* - This is a form with some predefined queries used to make common statistics on customers and on their behaviors;
- *Customers (Utenti)* - This form makes it possible to visualize and modify customers' data, to add new customers, to formulate offers, and so on.
- *Time Table (Palinsesto)* - This form makes it possible to change the time table, either by modifying courses' day and time or by modifying the instructor assigned to a course.
- *Discounts (Sconti Amm)* - This is a static form that shows the discount rates that are applied, at present, in the Sport Centre.



Fig. 5.1 The navigation form

We conclude this introductory section remembering that a navigation form is a form that contains other sub-form, which can be opened using the bookmarks placed at the top (as in Fig. 5.1) or at the left side of the navigation form.

Also, in order to create a navigation form it is necessary to:

- Create the sub-form (as standard form) first;
- Create a blank navigation form (from Create → Form → Navigation);
- Place the sub-forms on the navigation form.

2. The Dashboard - Creating charts based on queries

This form contains a set of saved queries that can be immediately executed clicking on the corresponding hyperlink.

Let us see some of these queries.

- *Q1 Memberships* - A parametric query, operating on groups, that counts the number of memberships, grouping them in terms of: Type {VIP, ..., Base}, Duration {Month, ... Annual} and Time Windows {Morning, Evening}.

```
SELECT Count(USERS.Id) AS Total
FROM MEMBERSHIPS INNER JOIN (DURATIONS INNER JOIN ([TIME WINDOWS] INNER JOIN
    CUSTOMERS ON [TIME WINDOWS].Id = CUSTOMERS.[Time Windows Id]) ON DURATION.Id =
    CUSTOMERS.DurationID) ON MEMBERSHIPS.ID = CUSTOMERS.MemberhsipID
HAVING MEMBERSHIPS.Name = [Insert Name] AND DURATION.Name= [Insert Duration] AND [TIME
    WINDOWS].Nome = [Insert Time Window]
```

- *Q2 Instructors' Number of Classes* - A selection query working on groups, that returns the total number of classes assigned to each instructor.

```
SELECT INSTRUCTORS.Name, INSTRUCTORS.Surname, Count(COURSES.ID) AS Total
FROM CORSI INNER JOIN (INSTRUCTORS INNER JOIN [COURSES-INSTRUCTORS] ON INSTRUCTORS.ID =
    [COURSES-INSTRUCTORS].InstructorId) ON COURSES.ID = [COURSE-INSTRUCTORS].CourseId
GROUP BY INSTRUCTORS.Name, INSTRUCTORS.Surname
```

- *Q3 Average Access' Rate* - A selection query, with a calculated field, that returns the total and the average number of accesses to the weight room, computed over a period defined by the user.

```
SELECT Count(ID) AS [Total Accesses], DateDiff('d',[From], Date()) AS [Within the last n° days], _
    Format([Total Access]/ DateDiff('d',[From], Date()), "Standard") AS [Average Accesses' Rate]
FROM ACESSES
WHERE ACESSES.Date >Format([From], "mm/dd/yyyy") AND ACESSES.Area = "[Weight Room]"
```

Note that DateDiff('d',[From], Date()) takes the difference, in number of days, from the current date and the date passed as input by the user.

Also note that Format([Total Access]/ DateDiff('d',[From], Date()), "Standard") computes the average access rate and format it as a standard percentage number.

- *Q4 Average Swimming Pool Access Rate* - A selection query identical to the previous one.
- *Q5 Average Time In* - A query based on a sub query that returns the average time spent by a customer inside the center.

```
SELECT Format(Avg(Minutes),"Standard") AS [Average Time In]
FROM (SELECT Datediff('m', [Time In], [Time Out]) AS Minutes FROM ACCESSES)
```

- *Q6 Total Number of Participants* - A selection query operating on groups, that shows, for each course, the total number of participants and that of absents, over the last X days.

```
SELECT COURSES.Name, RESERVATIONS.[Class Date], Count(RESERVATIONS.[ClassID]) AS Participants,
      Sum(RESERVATION.State) AS Registered, Nz(Sum(RESERVATIONS.[Absents]),0) AS Absents
FROM COURSES INNER JOIN RESERVATIONS ON COURSES.ID = RESERVATIONS.[COURSES.ID]
WHERE DateDiff('d',[RESERVATIONS].[Class Date],Date()) <= Cint([Number of day]) AND
      DateDiff('d',[RESERVATIOSN].[Class Date],Date())>=0
GROUP BY COURSES.Name, RESERVATIONS.[Class Date]
```

- *Q7 Average number of participants* - A selection query, operating on groups, that returns the average number of people that joined a course. The average is computing considering classes that were held starting from a certain data (indicated by the customer). Also note that the query is based on Q7b, another selection query that returns the number participants to each class.

```
SELECT Name AS Course, Format(Avg(PeopleIn), "Standard") AS [Average Number]
FROM Q7b
WHERE [Class Date] >= [Insert starting date]
GROUP BY Name
```

```
SELECT COURSES.Name, RESERVATIONS.[Class Date], Count(RESERVATIONS.[CourseID]) AS PeopleIn
FROM COURSES INNER JOIN RESERVATIONS ON COURSES.ID = RESERVATIONS.[CourseID]
GROUP BY COURSES.Name, RESERVATIONS.[Class Date]
```

In the same form there are also three hyperlinks that open three charts based on queries. We remember that, in order to create a plot based on a query the following steps are needed:

- Create and save a selection query that returns the data needed to create the chart. For instance, to create a bi-dimensional X-Y line chart, the query must return a table with (at least) two columns containing, respectively, the X and the Y values to be plotted;
- Create a blank form and, operating in design view, place a chart object onto it;
- Connect, following the wizard's instruction, the chart object to the query containing the input data;
- Complete the chart, selecting the fields containing the data and choosing an appropriate chart type.

The first hyperlink, namely *Chart 1 - Memberships Distribution*, opens the bar chart of Figure 5.2, which shows the distribution of the different kinds of membership cards, purchased by the customers over the last years. In this case the chart is based on the following selection query (operating on groups), which returns the total number of memberships grouped by typology.

```
SELECT Count(CUSTOMERS.ID) AS Total, MEMBERSHIPS.Name
FROM MEMBERSHIPS INNER JOIN CUSTOMERS ON MEMBERSHIPS.ID = CUSTOMERS.[MembershipID]
GROUP BY MEMBERSHIPS.Name
```

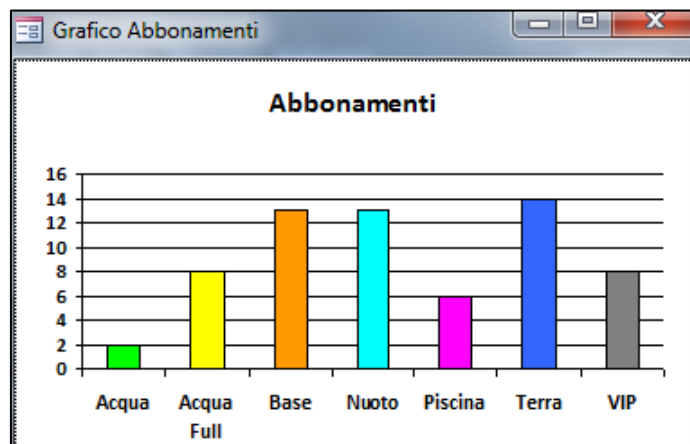


Fig. 5.2 The bar chart of the memberships' types

Clearly, in this case, after the association (between the form and the query) has been made, the following fields have been selected to create the bar chart:

- Total has been used to create the bars;
- Name has been used to create the labels of the bars.

The second hyperlink, namely *Chart 2 - Daily Accesses*, opens a bi-dimensional X-Y linear chart, which shows the trend of the daily accesses to the weight room and to the swimming pool, respectively.

The chart is based on a selection query (operating on group) that returns the total number of accesses, grouped by date and by area.

```
SELECT Count(ID) AS Accesses, DateIn, Area
FROM ACCESSES
GROUP BY DateIn, Area
```

The third and last hyperlink, namely *Chart 3 - Hearth Classes Day and Night*, opens a line chart showing the trend of the number registrations, in a month, to classes held at lunch time and in the evening, respectively. It is interesting to note that the query groups data using both the reservation data and the class time. Thus, data returned by the query are aggregated on a daily basis (i.e., by date); conversely, the final aggregation made on a monthly basis was made operating directly on the chart. The SQL code is shown below:

```
SELECT RESERVATIONS.[Class Date], ResType(RESERVATIONS.[Class Time]) AS Period, _
                                         Count(RESERVATIONS.ID) AS PeopleIn
FROM COURSES INNER JOIN RESERVATIONS ON COURSES.ID = RESERVATIONS.CourseID
WHERE (COURSES.Category ="Earth Courses") AND (IsNull([RESERVATIONS].[Absent]))<>False)
GROUP BY ResType(RESERVATIONS.[Class Time]), RESERVATIONS.[Class Date]
```

This query takes advantage of a public function, namely ResType, which associates the string containing the “Time Windows” of a class that corresponds to the value of the “Class Time” field, returned by the query.

The VBA code is shown below.

```
Public Function ResType(H As Date) As String
    If H <= #3:00:00 PM# And H >= #12:00:00 AM# Then
        ResType = "Lunch Time"
    ElseIf H >= #7:00:00 PM# And H <= #11:00:00 PM# Then
        ResType = "Evening"
    Else
        ResType = "Others"
    End If
End Function
```

3. The Customers Form

This form is dynamically and bi-directionally connected (i.e., Dynaset) with the CUSTOMERS table; thus to assure data integrity, some navigation and filtering buttons have been added at the bottom of the form. ù

3.1 Basic Buttons - Using the Wizard to add standard macro to navigation buttons

Buttons added on the form are the following ones:

- *Cancel* - All modifications are cancelled;
- *Save* - All modifications are saved in the Data Base;
- *Delete* - This button deletes a single record (i.e., a customer), provided that this customer is not linked to fields belonging to other tables. In other words, only new customers that have not used the services of the center yet, or customers that have been added by mistake can be deleted;
- *Filter by ID* - This button makes it possible to get customer's information, by specifying his or her ID;
- *Remove Filter* - This button removes the filter so that all customers are shown again on the screen.

The first three buttons were created using a wizard, thus their On Click event does not trigger a VBA procedure but, rather, it triggers an embedded macro. We recall that, in order to use the wizard the following steps are needed:

- The form must be opened in Design View;
- A command (such as a Button) must be selected from the object library;
- Once an object has been selected (from the Combo Box of the Design Bar) the option "Create through Wizard" must be selected;
- The selected command must be placed on the form;
- Following the wizard procedure a macro is automatically created and assigned to the command.

For instance, the Cancel button is associated to the Macro shown in Figure 5.3.

```

SuErrore
    Vai a Successivo
    Nome macro
EseguiComandoMenu
    Comando AnnullaRecord

If [MacroError]<>0 Then
    FinestraMessaggio
        Messaggio =[MacroError].[Description]
        SegnaleAcustico Sì
            Tipo Nessuno
            Titolo
End If

```


Fig. 5.3 *The Embedded Macro*

Conversely, the Filtering button uses a standard VBA code, as shown below.

```

Private Sub BtnFilterOn_Click()
Dim ID As Integer
Dim F As String
On Error GoTo Err
    ID = InputBox("Insert user Id", "ID Filter")
    F = "[ID] = " & ID
    If Not IsNull(DLookup("ID", "CUSTOMERS", F)) Then
        Me.Filter = F
        Me.FilterOn = True
    End If
Err:
    If Err.Number <> 0 Then
        Exit Sub
    End If
End Sub

```

Some standard buttons have also been added to recreate a navigation bar. For instance, in order to open a new and blank record the following button  must be used.

3.2 Additional Buttons

The form also contains three additional buttons:

- Show Existing Offers;
- Show Existing Quotations;
- Show New Quotations.

Specifically, the first one allows the administrator to watch the list of all the offers made to a customer. Also, as default, a filter is used to show only the offers relative to the customers selected in the Customers form. However, this filter can be deactivated, by operating on the filtering button placed on the form. The VBA code, associated to the On Loading event of the form is a standard one and it is shown just for the sake of completeness.

Just note that the form is opened on the last record of the form using the following line of code:

```
DoCmd.GoToRecord acDataForm, "OFFERS", acLast
```

```
Private Sub BtnShowOffers_Click()  
Dim WCond As String  
Dim IdOffer As Integer  
If Not IsNull(Me.ID) Then 'ID is the Customer ID shown on the CUSTOMER form  
    WCond = "UserID = " & Me.ID  
    IdOfferta = Nz(DLookup("[OfferID]", "OFFERS", WCond), 0)  
    If IdOfferta > 0 Then ' There is at least one offer  
        DoCmd.OpenForm "OFFERS", acNormal, , WCond  
        DoCmd.GoToRecord acDataForm, "OFFERS", acLast ' Go to the last record  
    Else  
        MsgBox "There are no offers", vbCritical, "Offers"  
    End If  
End If  
End Sub
```

The second command allows the administrator to watch all quotations that were made to a customer and that are still valid. The code is like the precedent one; the only difference concerns the filtering condition that must act also on the date. This is shown below:

```
WCond = "[ID Cliente] = " & Me.ID & " AND [Scadenza] >= " & Format(Date, "\#mm\dd\yyyy\#")
```

The third command is much more interesting and it may be used when a new customer goes to the reception desk with a printed copy of a valid quotation. In this case, it is sufficient to indicate the User's ID and the information system pre-fills (with customer's data) the form needed to add the new customer.

The VBA code of this button, based on a Recordset Clone, is shown below.

```
Private Sub CmdNewQuotationsi_Click()  
Dim Frm As Form ' An object variable that points to a form  
Dim ID As Integer  
Dim MySQL As String, Cond As String  
Dim Db As Database  
Dim Rs As DAO.Recordset, Rs2 As DAO.Recordset
```


On Error GoTo Err

ID = InputBox("Insert Quotation ID", "Quotation") ' This opens an Input Box

Cond = "[ID] = " & ID & " AND [Expiration Date] >= " & Format(Date, "\#mm\dd\yyyy\#") & "

Cond = Cond & "AND [CustomerId] Is Null" ' Date Validity and new customer (i.e., ID is Null)

If Not IsNull(DLookup("ID", "QUOTATIONS", Cond)) Then ' Check the existence of the quotation

MySQL = "SELECT * FROM QUOTATIONS WHERE " & Cond

Set Db = CurrentDb

Set Rs = Db.OpenRecordset(MySQL)

Set Frm = Forms![ADMIN HOME PAGE]![CUSTOMERS SUB FORM].Form ' The form variable

Set Rs2 = Frm.RecordsetClone ' A clone of the Form's Recordset

' Some fields are automatically pre-filled

With Rs2

.AddNew

.Fields("Name") = Nz(Rs.Fields("Name"), "")

.Fields("Surname") = Nz(Rs.Fields("Surname"), "")

.Fields("Date of Birth") = Nz(Rs.Fields("Date of Birth"), Date)

.Fields("Address") = Nz(Rs.Fields("Address"), "Street")

.Fields("Phone") = Nz(Rs.Fields("Phone"), 0)

.Fields("Email") = Nz(Rs.Fields("Email"), "@")

.Fields("Wellness") = Nz(Rs.Fields("Wellness"), False)

.Fields("Price") = Nz(Rs.Fields("Price"), 500)

.Fields("Starting Date") = Date

.Fields("MembershipID") = Nz(DLookup("ID", "MEMBERSHIPS", "[Name] = " _
& Rs.Fields("Membership Type") & ""), 1)

.Fields("Time Window ID") = Nz(DLookup("ID", "[TIME WINDOWS]", "[Name] = " _
& Rs.Fields("Time Window") & ""), 1)

.Fields("DurationID") = Nz(DLookup("ID", "DURATIONS", "[Duration] = " & Rs.Fields("Duration")), 1)

.Update

.Close

End With

Frm.Recordset.MoveLast ' We move to the last record of the form

Set Frm = Nothing

Else

MsgBox "Quotation not found", vbCritical, "Missing Quotation"

End If

Err:

If Err.Number <> 0 Then

Exit Sub

End If

End Sub

Note the way in which the current form is assigned to the Frm variable:

```
Set Frm = Forms![ADMIN HOME PAGE]![CUSTOMERS SUB FORM].Form ' The form variable
```

Specifically we have that:

- Forms is the collection of all the forms;
- ADMIN HOME PAGE is the name of the Main form;
- CUSTOMERS SUB FORM is the name of the sub form (property of the Main form);

Also note that the sub form is linked to the Customer form that, in turn, is linked with the CUSTOMERS table.

4. Form Time Tables

This form allows administrators to make changes to the time table of the sport center. The important thing is that the system assures the congruity of the couple “Course - Instructor” i.e., only instructors that have the right qualification can be assigned to a course.

More specifically:

- If the user wants to change a course, the list of the instructors that can be assigned to that course is automatically shown;
- If the user wants to change the instructor assigned to a course, the list of the assignable instructor is automatically displayed.

For example, as shown in Figure 5.4, selecting from the Combo Box a course different from the original one, the “Add Instructor” form is automatically displayed and the user can select an appropriate instructor for the course. The same form is automatically open even if the user tries to modify an instructor.

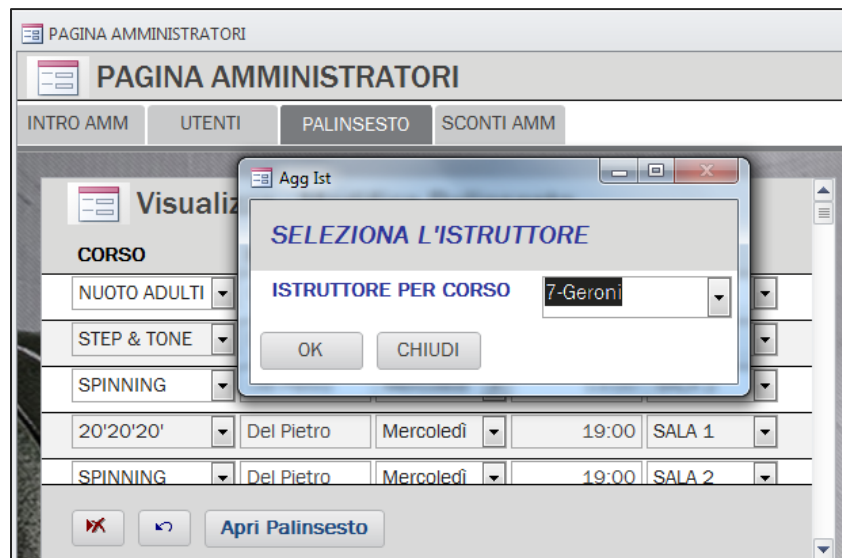


Fig. 5.4 Time Table and Add Instructor Forms

To this aim, any time a value changes either in the Combo box of the courses or in the Combo box of the Instructors, the Add Instructor form is automatically displayed. The code, triggered by the After Update event of bot Combo Box is thus:

```
DoCmd.OpenForm "Add Instructor"
```

Obviously, data (i.e., the list of qualified instructors) are loaded on the Add Instructor form when the form is loaded.

Please note that, in this case, we have two concatenated sub forms:

- The main form [ADMIN HOME PAGE] contains the sub form [CUSTOMERS SUB FORM];
- The sub form [CUSTOMERS SUB FORM] contains a second sub form called [Time Table SUB FORM]

For this reason, if we wanted to reference the last sub form, the following path should be used:

```
Forms![ADMIN HOME PAGE]![CUSTOMERS SUB FORM]![TIME TABLE SUBS FORM].Form
```

The full code is shown below.

```
Private Sub Form_Load()  
Dim MySQL, Instructor As String  
Dim Pos As Integer  
Dim Db As Database  
Dim Rs As DAO.Recordset  
' We write a query that returns all the instructors that can be assigned to the course corresponding to the ID  
' selected on the Time Table Sub Form. Instructors are ordered by Priority  
MySQL = "SELECT INSTRUCTORS.ID, INSTRUCTORS.Surname, [COURSES-INSTRUCTORS].[Principal Instr], _  
        [COURSES-INSTRUCTORS].[Course ID]"  
MySQL = MySQL & " FROM INSTRUCTORS INNER JOIN [COURSES-INSTRUCTORS] ON _  
        INSTRUCTORS.ID = [COURSES-INSTRUCTORS].[Instructor ID]"  
MySQL = MySQL & " WHERE [COURSES-INSTRUCTORS].[CourseID] = " & _  
        Forms![ADMIN HOME PAGE]![CUSTOMERS SUB FORM]![TIME TABLE SUBS FORM].Form![CourseID]  
MySQL = MySQL & " ORDER BY [COURSES-INSTRUCTORS].[Principal Instr] ASC" ' This is the opening query  
Set Db = CurrentDb  
Set Rs = Db.OpenRecordset(MySQL)  
Pos = 0  
' We cycle on all qualified instructors  
Do While Not Rs.EOF  
    Instructor = ""  
    Instructor = Rs.Fields(0) & "-"  
    Instructor = Instructor & Rs.Fields(1) ' ID and name of the instructor are combined  
    Me.CmbInstructor.AddItem Instructor, Pos ' A new instructor is added to the list in a different position  
    Rs.MoveNext  
    Pos = Pos + 1  
Loop
```

```
Me.CmbInstructor.Value = Me.CmbInstructor.ItemData(0) ' The principal instructor (in position 0) is shown
Rs.Close
Set Db = Nothing
Set Rs = Nothing
End Sub
```

The OK button confirms the modification. To this aim:

- It splits the ID from the surname and writes the ID in the CmbInstructor Combo Box of the TIME TABLE SUB form.
- Since this sub form is dynamically linked to the CURSES-INSTRUCTORS bridge table, the modifications are immediately registered on the Data Base.

The Close button, conversely, closes the Add Instructor Form and aborts any previous modifications made on the TIME TABLE SUB form. So, even if the user had changed a course, the original one (and the original instructor) is placed back on the form.

The full code is shown below.

```
Private Sub ComOk_Click()
Dim ID As Integer
Dim Surname As String
Dim Str () As String
Str = Split(Me.CmbInstructor.Value, "-")
ID = Str(0)
Surname = Str(1)
' write the ID in the sub form
Forms![ADMIN HOME PAGE]![CUSTOMERS SUB FORM]![TIME TABLE SUBS FORM].Form.![InstructorID] = ID
DoCmd.Close
End Sub

Private Sub CmdEsc_Click()
Forms![ADMIN HOME PAGE]![CUSTOMERS SUB FORM]![TIME TABLE SUBS FORM].Form.Undo
DoCmd.Close
End Sub
```