Contents Page

Preface

The purpose of this Trainer iii
Summary of topics from 'Getting Started with VBA' Trainer iii
Version of Access used iii
Other sources of information iii

PART 1 – PASSWORD PROTECTION 1

1.1 Introduction 1
1.2 Using the Access built-in password feature 2
1.3 Access user level security 3
1.4 Setting up your own password file 4
1.4.1 Creating a password database and linking it to your application 4
1.4.2 Creating a login form 5
1.4.3 Checking the log-in ID and password 6
1.4.4 Testing the code 8
1.4.5 Encrypting the database 8
1.5 Using network log-in information 9
1.6 Exercises 10
1.6.1 Limit the number of tries 10
1.6.2 Allow for case-sensitive passwords 10
1.6.3 Allow the user to change their password 11
1.6.4 Password control application 11
1.6.5 Combine login form values and network login information 11
1.6.6 Put network login check in Main Menu 11
1.6.7 Use access levels to restrict user's view of data 12
1.6.8 Record who made data changes 12
1.6.9 Keep a User Log 12

PART 2 – ERROR TRAPPING AND CUSTOM ERROR MESSAGES 13

2.1 Introduction 13
2.2 Error Codes 14
2.3 Coding On Error within a procedure and using the Err Object 14
2.3.1 Display your own message in place of an Access message 14
2.3.2 Suppress an unwanted Access information message 15
2.3.3 Checking for required/mandatory fields 15
2.4 Using the Form_Error event 16
2.4.1 Checking a relationship in another table 16
2.5 Exercises 17
2.5.1 Using public constants for error codes 18
2.5.2 Own message for start/end of a set of records on a form 18
2.5.3 Trapping missing fields when moving to next/previous records 18

PART 3 – USING DATA ACCESS OBJECTS (DAOs) 19

3.1 Introduction 19
3.1.1 Data Access Objects (DAO) 19
3.1.2 ActiveX Data objects (DAO) 19
3.1.3 Some references for further information 20
3.1.4 Using DAO code in Access 2000+ 20
3.2 Basics of DAO code 22
3.2.1 Defining a Database Object 22
3.2.2 The Execute method 22
3.2.3 A very simple example to start with 23
3.2.4 Recordset Object, methods and properties 24
3.2.5 A more complex example 25
3.3 Adding to a combo box at run-time 28
3.4 Creating your own Domain Aggregate functions 29
3.4.1 myDCount function – version 1 29
3.4.2 myDCount function – version 2 31
3.4.3 myDLookup function – version 1 32
3.4.4 A common procedure for the myD functions 33

Continued over…

Contents Page

3.5 The Object Browser 35
3.6 Summary 36
3.7 Exercises 36
3.7.1 Create tables for bookings this year, split by type 36
3.7.2 More myD functions 36

PART 4 – WORKED EXAMPLE OF BOOKING PROCEDURE 37

4.1 Introduction 37
4.2 Create a dialog box to get the booking date 38
4.3 Creating a BookingGrid form 39
4.3.1 Declaring variables 39
4.3.2 Initialising the arrays 41
4.3.3 Stage 1 - get the booking table records 42
4.3.4 Stage 2 - read Recordset details and put into arrays 43
4.3.5 Stage 3 - write arrays out to a new table 46
4.3.6 Stage 4 - create and use the BookingGrid form 47
4.3.7 Delete the temporary table 48
4.4 Make a booking 49
4.4.1 Create and use textboxes for parameter values on the BookingGrid form 49
4.4.2 Create a Bookings form and code 50
4.5 Check/Delete a booking 52
4.6 Some improvements that could be made 54
4.7 Diagram showing how all this Part fits together 55
4.8 Booking for fractions of an hour 55
4.9 Exercises 56

Index 57

PREFACE

The purpose of this Trainer

This document follows on from the ‘Getting Started’ VBA Trainer, version 5.1. That document showed techniques that most students should be able to use to improve their MS Access applications, even if they are not that fond of programming.

This ‘Further VBA’ Trainer introduces some more advanced VBA coding, including DAO code using Database and Recordset objects, and assumes that the reader will already be familiar with the contents of the ‘Getting Started’ VBA Trainer. This second Trainer follows the same format as the previous Trainer, and assumes that readers will work through the examples given in each section and attempt the exercises. Later examples and exercises may require that earlier ones have been done.

The document further develops the basic Chelmer Leisure scenario, taking it from the point reached at the end of the earlier Trainer (but it is not necessary to have a version split into front/back-ends).

Macros are not used in this document, as Microsoft literature has stated that it is intended that VBA replace macro use.

Summary of topics from ‘Getting Started with VBA’ Trainer

The topics illustrated include:

- Overview of VBA coding basics
- Code modules
- The Debugger
- Creating own procedures (such as myDisplayInfoMessage, myYesNoQuestion).
- Built-in functions.
- Event code on forms for data maintenance processes
- Menus and sub menus
- The Forms Collection
- Automatic calculations on forms
- Validation of data entry on forms
- Searching for, and filtering, records on forms
- Event code for combo and list boxes
- Event code on reports
- Embedded SQL using DoCmd.RunSQL
- Tabbed forms
- Splitting a database into front-end and back-end, and preparing your application for distribution.
- Three worked examples of methods that can be used for making bookings

Version of Access used.

MS Access 2002, under MS Office XP, with Access 2000 database format.

Other sources of information

http://www.cse.dmu.ac.uk/~mcspence/Access.htm for FAQs, example databases, further links, etc.

A useful book for further reading (referred-to as *Robinson* in this Trainer) is:

Real World Microsoft Database Protection and Security
Garry Robinson, published by APress
ISBN 1-59059-126-7
£43.00 (in autumn 2004)
*Reviewed in BCS Computer Bulletin November 2004. Star rating **** (good)*

Copies of this book should be in the Kimberlin Library.

PART 1 – PASSWORD PROTECTION

REVIEW OF PART 1:

In this part of the Trainer you will see…

- … that creating appropriate security can be a complex process, in which you need to take many factors into account
- … that, if password protection is required, it may be best done last, after the rest of the application is working (at least for projects and assignments)
- … that Access has its own simple password protection for a database
- … that Access also has a much more complex security feature for users and workgroups
- … how to set up your own password check and set levels of user access
- … that Access can encrypt a database
- … how to access the user's network log in name

1.1 Introduction

Many students like to add password protection to their Projects, mainly, it seems, because they think it might be useful. Well, yes, but you need to think also about the environment in which the database will be used. The need for, and type of, password protection may well depend upon the application and the conditions under which it would be run. If the user had to log on to a networked machine then (possibly?) all installed software on that machine would have been approved for that user to use.

Companies may well have their own security procedures to which all applications must adhere. If your application required, as a basic requirement, for users to have different levels of access to data and functions, then you may need to have something in place that would link log-in details with data and function access levels.

The back-end database may also be on a server that could only be accessed by authorised users, so a separate password procedure may not be needed for a particular application.

You should also remember that, unless security is a vital requirement of your system, getting the main functions of the system working is (normally) more important than creating password protection. Password protection can be an add-on once the main system has been completed, at least for Projects.

If you are developing a Project, then discuss password protection with your supervisor before you attempt to implement it. Unless the Project is required for an external user, and/or password protection is a vital part of the required functionality, you may well be advised to concentrate on the functionality in preference to adding password protection.

However, if you really want to set password protection to your database, this section discusses some possible ways for you to achieve it. You should note that hard-coding a password in your code is **not** a valid method!

Look at MS Access *Help* with the keyword *security* for further information regarding securing a database.

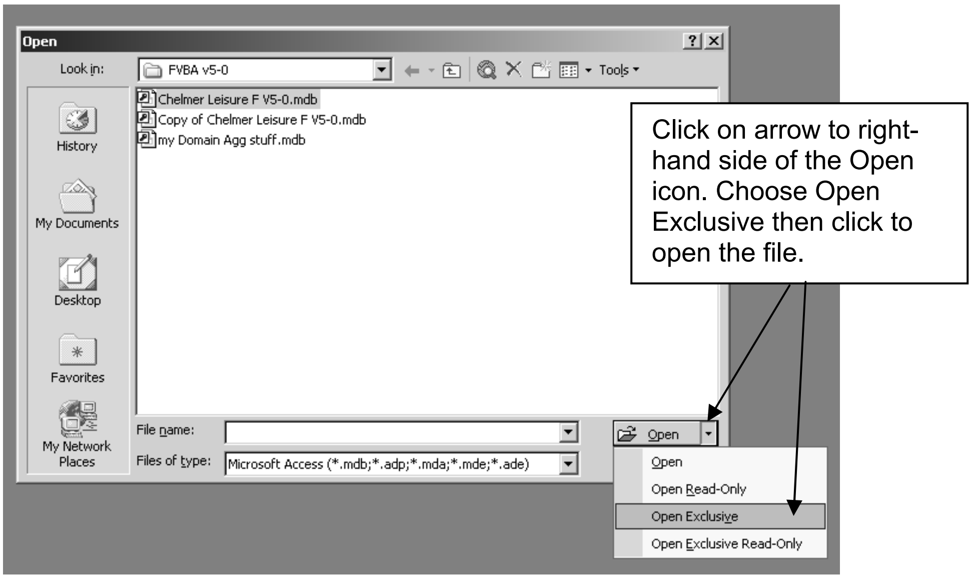
1.2 Using the Access built-in password feature

This method could be useful if your database is in an open office and/or you sometimes leave your machine unattended after you have logged on, or if you (and thus anyone else) can use the machine without needing to log on. If you leave your machine unattended for lengthy periods of time, it is probably best to log off or lock your machine. To see how to lock your computer use the Windows *Help and Support Center* facility with '*lock your computer*'.

Access has a built-in password feature which is very simple to use, but needs the Tools menu to be on the application menu bar. Passwords are not much good if they are known to several people (unless the passwords are changed regularly) so this method is probably best used for an application (or the front-end of an application) that resides on your own machine. An application (or a back-end database) on a shared network area may be able to use this method (controlled by a database administrator, who changes the password and notifies users), but a shared application may need a more appropriate and safe method than this.

Do the following to set the password:

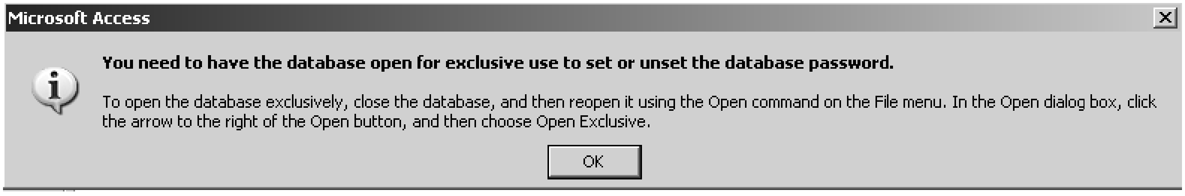
- Open your database (or front-end application) using *File→Open* and open it for exclusive use (see Fig 1.2.1).



Click on arrow to right-hand side of the Open icon. Choose Open Exclusive then click to open the file.

Fig 1.2.1 Opening a database via File→Open, and specifying exclusive use.

- Choose *Tools→Security→Set Database Password* from the database menu. Enter and verify your password. You will get the error message shown in Fig 1.2.2 if the database is not opened in exclusive mode.


Microsoft Access

You need to have the database open for exclusive use to set or unset the database password.

To open the database exclusively, close the database, and then reopen it using the Open command on the File menu. In the Open dialog box, click the arrow to the right of the Open button, and then choose Open Exclusive.

Fig 1.2.2 Message if you attempt to set a password for a database open in shared mode.

- Close your database and reopen it. You will be asked to enter your password first. See Fig 1.2.3.

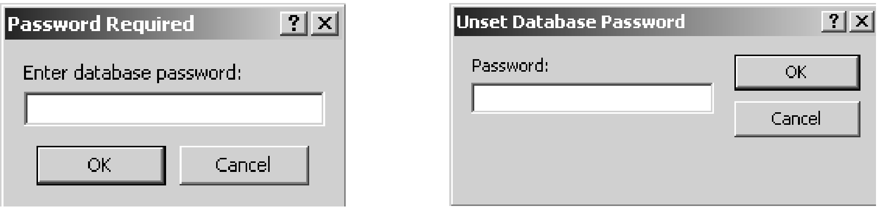


Fig 1.2.3 Dialog box asking for password when you open the database (on left) and when you want to unset the password (on right)

- To remove the password, open your database (and enter the password), and choose *Tools→Security→Unset Database Password* from the Database menu. You will be asked to confirm the password again. See Fig 1.2.3.
- To amend the password you first remove it then reset it.
- Don't forget the password! Access Help warns *"If you lose or forget your password, it can't be recovered, and you won't be able to open your database."*

1.3 Access user level security

Access provides a method of setting advanced security which may, or may not, be what you want. The version prior to Access 2000+ looks to be fairly complicated. Indeed, Evan Callahan (page 250 *Microsoft Visual Basic Step by Step*, 1995) says:

> *"Beware, however: advanced security is not for the faint of heart. Although Microsoft Access provides all the tools you need for setting up security it is a time-consuming and challenging process".*

Help for Access 2000 said (if I remember correctly) that this feature had now been improved and simplified.

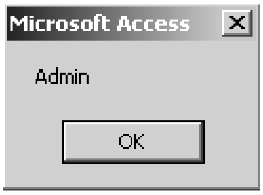
From my reading of various instructions regarding this feature, it is not obvious whether it would allow flexible permissions for users using different levels of access for types of data, as may be required for an application. For example, you may want all staff to be able to have read/write access to a table (via a form), but to restrict what fields of the table could be seen according to the user's access level; Access user level security may not be able to do that. However, you can use the CurrentUser method to see the user name:

```
MsgBox Application.CurrentUser
```

or

```
MsgBox CurrentUser
```

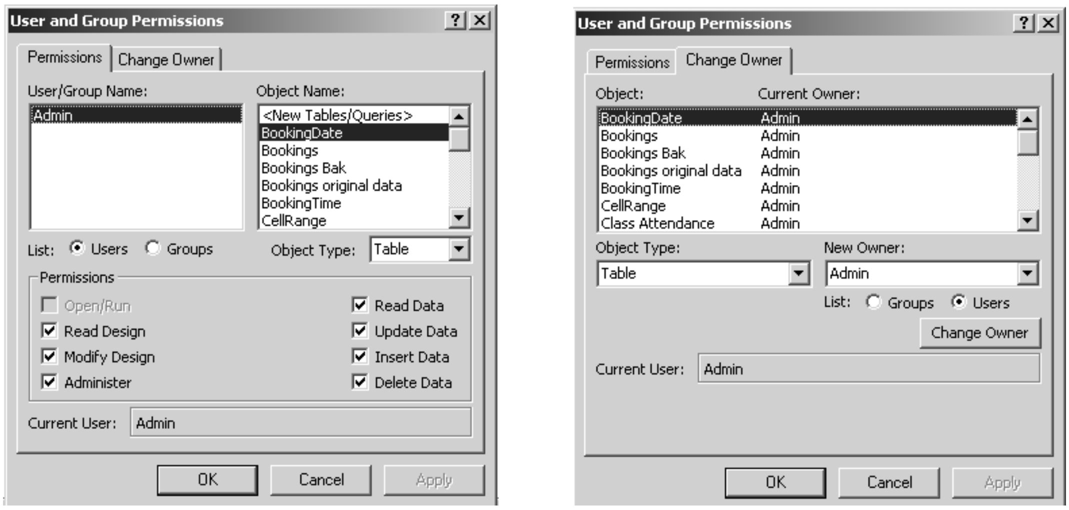
See Fig 1.3.1. Note that Admin is the default user name.



Admin

Fig 1.3.1 seeing who the current user is

It looks as though permissions have to be set for each object in a database; see Fig 1.3.2. The dialog box shown is obtained via *Tools→Security→User and Group Permissions* from the Database menu.



User/Group Name: Admin

Object Name: <New Tables/Queries>, BookingDate, Bookings, Bookings Bak, Bookings original data, BookingTime, CellRange

List: ● Users ○ Groups

Object Type: Table

Permissions:
□ Open/Run
☑ Read Design
☑ Modify Design
☑ Administer
☑ Read Data
☑ Update Data
☑ Insert Data
☑ Delete Data

Current User: Admin

Object:	Current Owner:
BookingDate	Admin
Bookings	Admin
Bookings Bak	Admin
Bookings original data	Admin
BookingTime	Admin
CellRange	Admin
Class Attendance	Admin

Object Type: Table

New Owner: Admin

List: ○ Groups ● Users

Current User: Admin

Fig 1.3.2 Setting user and Group permissions

Chapter 10 of *Robinson* discusses this feature in some detail.

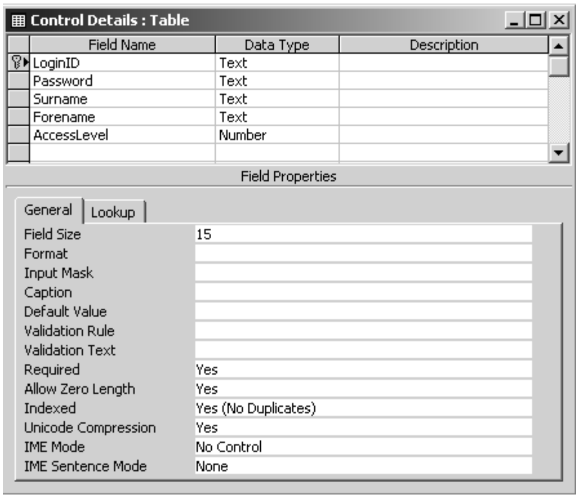
I will leave you to explore this feature further, should you wish to. This may be the most secure of the methods discussed here of protecting an application and its data. By combining this feature with the CurrentUser method, you may be able to get a flexible method of controlling access to fields in table.

1.4 Setting up your own password file

1.4.1 Creating a password database and linking it to your application.

Create a new database, called **CL Password Control** with just one table called **Control Details**. All fields have Required = Yes. See Fig 1.4.1.

Field name	Datatype	Size	Other
LoginID	Text	15	Primary key
Password	Text	25	
Surname	Text	30	User’s family name
Forename	Text	20	User’s first name
AccessLevel	Number	Byte	0 = no access (default) 1 = full access 2 = view only etc … as required by the application(s)



Control Details : Table

Field Name	Data Type	Description
LoginID	Text	
Password	Text	
Surname	Text	
Forename	Text	
AccessLevel	Number	

Field Properties

General | Lookup

Field Size	15
Format	
Input Mask	
Caption	
Default Value	
Validation Rule	
Validation Text	
Required	Yes
Allow Zero Length	Yes
Indexed	Yes (No Duplicates)
Unicode Compression	Yes
IME Mode	No Control
IME Sentence Mode	None

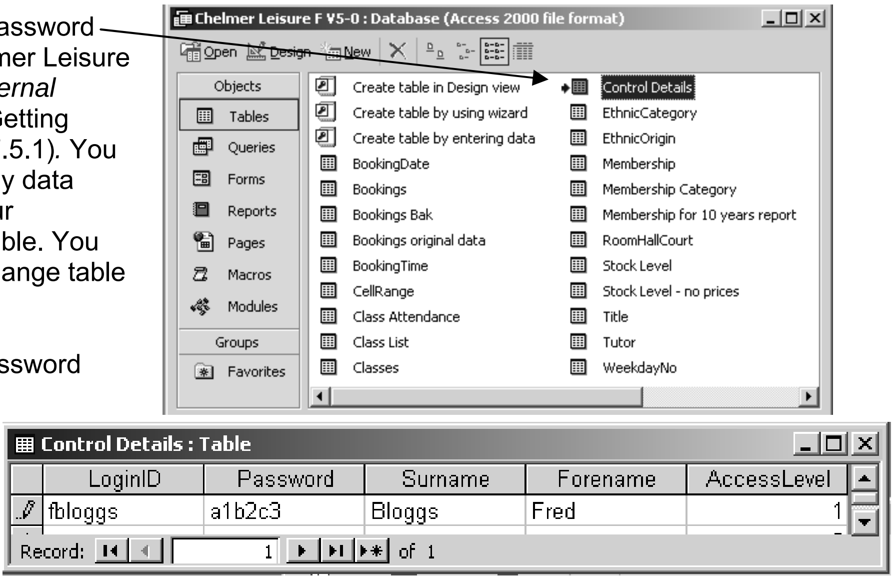
If you wanted to use the table for several applications, then you may need an AccessLevel field for each application.

Fig 1.4.1 Control details table for password database.

I have specified a separate table here so that it can be kept in a secure area and controlled separately. Only specified personnel would be able to add users to the database or amend existing data. See exercise 1.6.4.

Now link the table from your password database into your main Chelmer Leisure database, using *File→Get External Data→Link Tables* (See the ‘Getting Started’ VBA Trainer section 7.5.1). You can open the table and see any data present, which means that your application can also use the table. You can update data but cannot change table design details.

Add details of a user to the password table, as shown in Fig 1.4.2.



Control Details : Table

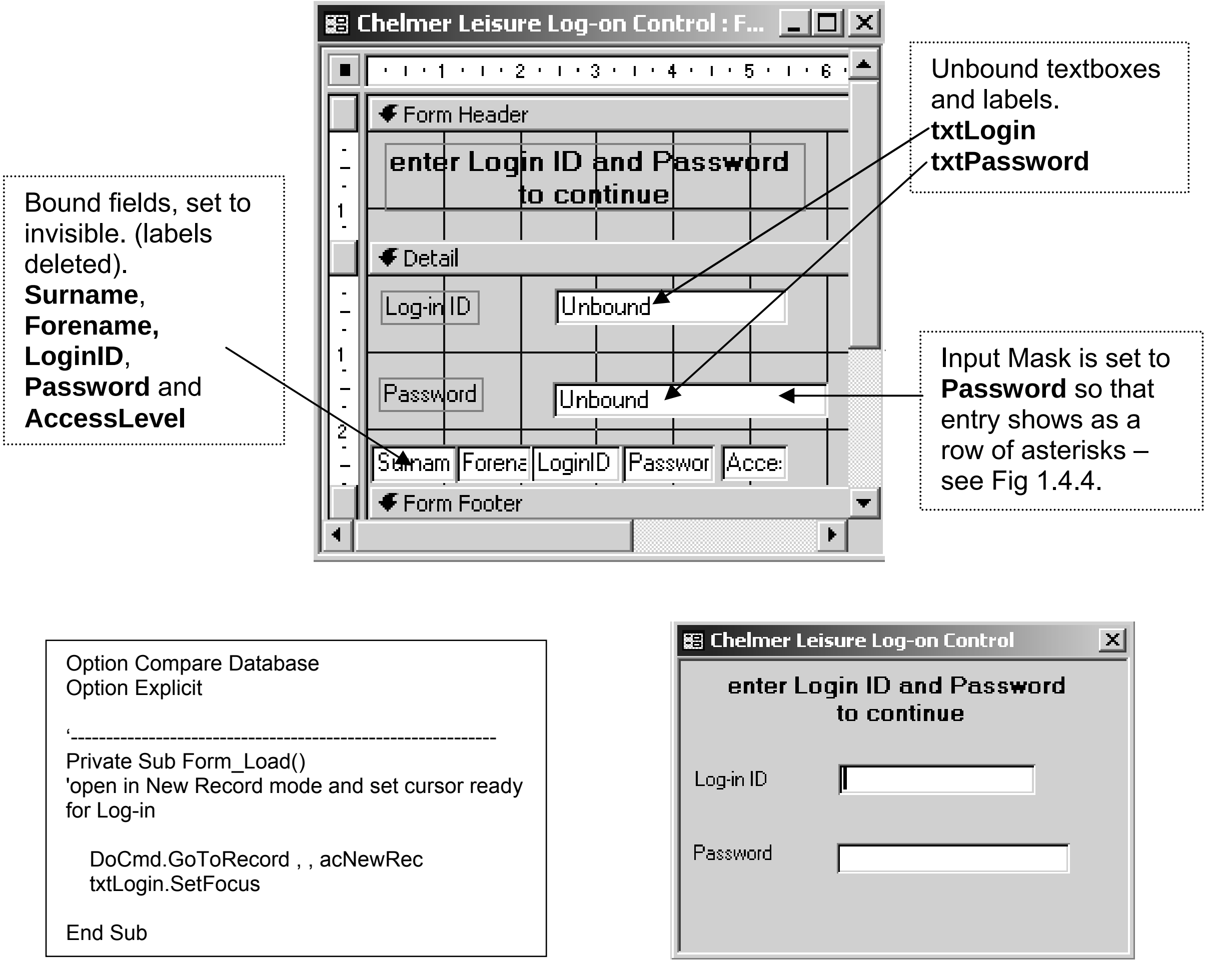
LoginID	Password	Surname	Forename	AccessLevel
fbloggs	a1b2c3	Bloggs	Fred	1

Record: 1 of 1

Fig 1.4.2 The Chelmer Leisure database window, showing the password table linked in.

1.4.2 Creating a login form

In your Chelmer Leisure database, create a columnar form with the title **Chelmer Leisure Log-on Control**, using all the fields from the **Control Details** table, as shown in Fig 1.4.3.


Chelmer Leisure Log-on Control : F...
Form Header
enter Login ID and Password to continue
Detail
Log-in ID
Unbound
Password
Unbound
Surnam
Forena
LoginID
Passwor
Acce:
Form Footer
Chelmer Leisure Log-on Control
enter Login ID and Password to continue
Log-in ID
Password

Unbound textboxes and labels. **txtLogin** **txtPassword**

Bound fields, set to invisible. (labels deleted). **Surname**, **Forename**, **LoginID**, **Password** and **AccessLevel**

Input Mask is set to **Password** so that entry shows as a row of asterisks – see Fig 1.4.4.

```
Option Compare Database
Option Explicit

'------------------------------------------------------------
Private Sub Form_Load()
'open in New Record mode and set cursor ready
for Log-in

    DoCmd.GoToRecord , , acNewRec
    txtLogin.SetFocus

End Sub
```

Fig 1.4.3 Log-on form in design view (top) and form view (bottom right) with initial code (bottom left)

Points to note:

- The form navigation controls, record selectors and scroll bars have been removed (use the form property box).
- The labels for the bound fields have been removed and the fields have been set to invisible and moved out of the way on the form.
- Two new unbound textboxes called txtLogin (label caption = Login ID) and txtPassword (label caption = Password) have been created, into which the user will enter the relevant information.
    - The input mask for txtPassword has been set to Password. Note that this does not check for case sensitivity; see exercise 1.6.2.
- The Form_Load event has code to open the form in new record mode with the cursor in txtLogin.

Your form should now look something like that shown in Fig 1.4.3 when it is first opened.

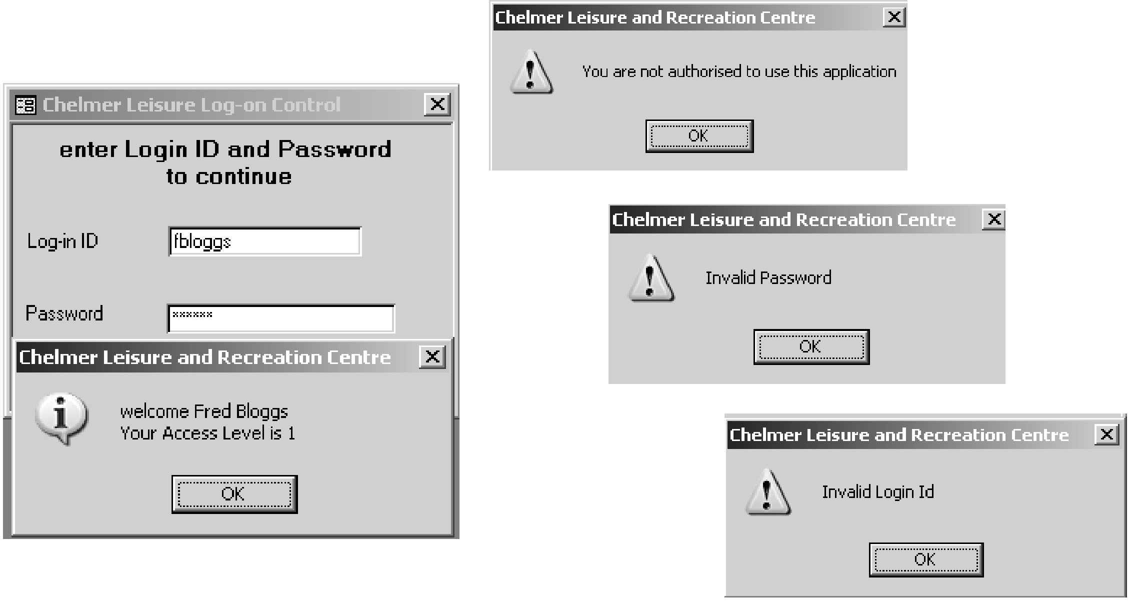
1.4.3 Checking the log-in ID and password

We now have a form into which the user can type their log-in details and a table against which we can check the details.

Create an After_Update event for txtPassword and enter the code shown in Figure 1.4.4.

A detailed explanation of the code is given on the following page. Use F1 to check *Help* for anything that you do not understand.

```
Private Sub txtPassword_AfterUpdate()
'this event is called when the user moves out of txtPassword

'first filter the record for the form using the LogInId (unique primary key)
'filter for matching login id
    DoCmd.ApplyFilter , "[LogInId] = forms![Chelmer Leisure Log-on Control]!txtLogin"

'if a match was found then the LoginID and Password from the table will be in the bound fields on the form.
'so these can be checked with what the user has typed in.
    If IsNull(LoginID) Then                    'no match for LoginID found
        myDisplayWarningMessage "Invalid Login Id"
    ElseIf Password <> txtPassword Or IsNull(txtPassword) Then         'invalid or missing password
        myDisplayWarningMessage "Invalid Password"
    ElseIf AccessLevel = 0 Then                'not allowed to use this application
        myDisplayWarningMessage "You are not authorised to use this application"
        DoCmd.Close             'close this form while testing
        'DoCmd.Quit             'to exitAccess in final version

    Else                                'valid log in, password and access level

       'say welcome - not essential, but useful testing aid
        myDisplayInfoMessage "welcome " & Forename & " " & Surname _
                                  & vbCrLf & "Your Access Level is " & AccessLevel

        DoCmd.OpenForm "Chelmer Leisure Main Menu"               'open the main menu
        Visible = False     'hide this form - code can access it to check AccessLevel

    End If

End Sub
```


Chelmer Leisure and Recreation Centre

You are not authorised to use this application

Chelmer Leisure Log-on Control

enter Login ID and Password to continue

Log-in ID: fbloggs

Password: ******

Chelmer Leisure and Recreation Centre

Invalid Password

Chelmer Leisure and Recreation Centre

welcome Fred Bloggs
Your Access Level is 1

Chelmer Leisure and Recreation Centre

Invalid Login Id

Fig 1.4.4 Code to check password details, message box when correct details are entered and the warning (error) message boxes

Explanation of the code in Fig 1.4.4:

- **DoCmd.ApplyFilter , "[LogInId] = forms![Chelmer Leisure Log-on Control]!txtLogin"**
  - See section 3.5 of the "Getting Started" VBA Trainer for information about filtering records on forms.
  - What the code here is doing is to filter the records for the form so that the record that matches the login typed by the user is found. The details will not show on the form as the bound fields are invisible, but the values in these fields can be used by code.
  - *Chelmer Leisure Log-on Control* is the name of my form.

- **If IsNull(LoginID) Then 'no match for LoginID found**
  - **myDisplayWarningMessage "Invalid Login Id"**
    - If no matching record was found, then the bound LoginID field contents will be Null.
    - The procedure myDisplayWarningMessage is from the "Getting Started" VBA Trainer.

- **ElseIf Password <> txtPassword Or IsNull(txtPassword) Then 'invalid or missing password**
  - **myDisplayWarningMessage "Invalid Password"**
    - If a matching record was found, then check to see if the password entered by the user matches that from the record for this LoginID, or if the user has entered a password at all.
    - Oddly, although one would think that the check If Password <> txtPassword would pick up the situation where txtPassword was Null, it doesn't do so! Hence the check for IsNull as well.

- **ElseIf AccessLevel = 0 Then 'not allowed to use this application**
  - **myDisplayWarningMessage "You are not authorised to use this application"**
  - **DoCmd.Close 'close this form while testing**
  - **'DoCmd.Quit 'to exit Access in final version**
    - Login and password match, but if AccessLevel = 0 then user is not authorised to use this application. The password database could control access to more than one application.
    - The login form is closed (in the final version Access could be exited).

- **Else 'valid log in, password and access level**
  - **'say welcome - not essential, but useful testing aid**
  - **myDisplayInfoMessage "welcome " & Forename & " " & Surname _**
    & vbCrLf & "Your Access Level is " & AccessLevel
  - **DoCmd.OpenForm "Chelmer Leisure Main Menu" 'open the main menu**
  - **Visible = False 'hide this form - code can access it to check AccessLevel**
    - Login and password match and user is authorised to use this application.
    - Code here displays a welcome message confirming the user's name and access level. This is not necessary but is useful for testing to check that you are picking up the right details. The AccessLevel would be more useful as a description ('Full' for example, in this case) rather than a number. One method could be to have the field as a combo box and use the Column property to pick up the description (see 'Getting Started' VBA Trainer section 3.6)
    - The procedure myDisplayInfoMessage is from the "Getting Started" VBA Trainer.
    - The application main menu is opened.
    - The login form is hidden (set to invisible) so that the user cannot see it. It is therefore still available for code modules to access and use the AccessLevel as appropriate to allow/restrict access to data or to use the log in as part of an audit trail of data access and changes.

- **End If**
  - End of the code.

Now you can make your login form open automatically on start up (use *Tools→StartUp* - see section 4.2.4 of the "Getting Started" VBA Trainer).

1.4.4 Testing the code.

When you open the database and enter the correct information in the login form, the screen should look like that in Figure 1.4.4, then control is passed to the main menu and the login form is hidden. If you enter an incorrect login or password then you should get the appropriate warning (error) message, the menu form is not opened and the login form stays open for the user to try again. If the login and password are correct, but the user is not authorised to use the application, then the warning message is shown and the login form is closed (or Access is exited, depending on which option you have chosen here).

Fig 1.4.5 shows a possible test plan for the login form.

Test No	**Data**			**Reason for test**	**Expected result**
	LoginID	**Password**	**Access Level**		
1	fbloggs	a1b2c3	1	Valid login and password for full access.	"Welcome Fred Bloggs. Your access level is 1." Main menu opens. Login form is hidden.
2	fbloggs	A1B2C3	1	As test 1 – checking effect of uppercase in password. *See also exercise 1.6.2.*	As test 1. Case differences are not significant.
3	Anything not in the Control details table	anything	----	Invalid login.	"Invalid Login ID". User stays with login form. Menu not opened.
4	fbloggs	aaaaaa	----	Invalid password.	"Invalid Password". User stays with login form. Menu not opened.
5	*New **	*New **	0	Valid login and password but user is not authorised for any access.	"You are not authorised to use this application". Login form (or application) closes.
6	Null	anything	----	User has left login empty.	As test 3
7	fbloggs	empty (enter & delete)	1	User has entered valid login but has left password empty.	As test 4

* add a new record to the password table for test 5

Fig 1.4.5 Possible test plan for the login form

Finally, it should not have escaped your attention that the developer can see the password details as the Password field on the Control Details table does not have an input mask.

But note that if you change the input mask on the Control details file for the Password field to Password then look at the table via the link in your Chelmer Leisure database you will see that the Password field shows as a string of asterisks. So far so good - the information looks to be secure. But move to design view and attempt to delete the input mask - Access will say you can't, OK, but if you change back to table view you can see the full password! If you attempt to save the change to the input mask, it will simply be ignored, but it has operated on a temporary basis until then.

You can also see the password via the Debugger, regardless of the value of the input mask.

So this information is not entirely secure. However, it is not unusual for developers to have access to sensitive information; the important thing is to prevent application users having access.

1.4.5 Encrypting the password database.

Access has an encryption feature that may be useful. See Access *Help* with the keyword *encryption*.

1.5 Using network log-in information

The user may have had to log in to the machine before being able to access any of the software. It could thus be an acceptable alternative to check this user information on the network against the password file, rather than requiring the user to enter login information and password again. This method will not require passwords, and also means that there is no further login required to access the application. It could thus be useful where a user does not leave a machine unattended for anyone else to use. See also exercise 1.6.5.

Make a second version of your Chelmer Leisure Log-on Control form and do the following:

- New module for password code:
  - Add the code shown in Fig 1.5.1 to a new Access Module.
- Copy of login form.
  - Change the code so that it looks like that in Fig 1.5.2.
    - All coding is now in the Form_Load event so is run automatically when the form is opened.
    - Replace the first two lines (opening with a blank form and positioning the cursor) in the Form_Load event with `txtLogin = myGetCurrentUser 'get user login id`
      This uses the new function from Fig 1.5.1.
    - Use the 'unauthorised' message for both invalid login and a zero access level, and remove all references to the password. The password column on the password database table could now be removed (unless it's also used in another application).
- Main menu.
  - In the Form_Load event, close the login form. Setting it to invisible it as before doesn't work as the login form hasn't finished loading at that stage. I've tried playing with various different events in both forms but can't work out how to set the login form to invisible after loading.
  - If you needed to keep the login and access level details for later reference, then copy them (using the Forms Collection – see 'Getting Started' VBA Trainer Appendix I) to hidden fields on the menu.

```
Option Compare Database
Option Explicit

'How to obtain the Log in Id of the current user
'Matthew Dean 2002
'www.cse.dmu.ac.uk/~mjdean

'declare function via the windows api using dll advapi31 (Advanced Windows 32 Base API)
'lpBuffer is a buffer that returns a null terminated string containing the user name
'lpsize states the size of the buffer used
'lpsize and the buffer size used must be the same
'if the buffer is too small the function will also fail
'the function returns true or false depending on if the operation works or not
Declare Function GetUserNameA Lib "advapi32.dll" (ByVal lpBuffer As String, lpsize As Long) As Boolean

'---------------------------------
Public Function myGetCurrentUser() As String
'function returns the name of the current user (at least on NT 4 it does!)

Dim BufferSize As Long          'used to control the size of the buffer
Dim UserName As String          'buffer to store the user name
Dim Success As Boolean          'used to flag if it works or not

    BufferSize = 255            'set buffer size to 255 char (should be long enough!)
    UserName = Space(BufferSize) 'initialise the buffer

    'make the api call populating buffer with the user name
    'must specify the size of the buffer used in 2nd parameter
    Success = GetUserNameA(UserName, BufferSize)

    'strip off unwanted characters
    'the api call populates the buffer with a null terminated string
    'so we can use the null value (0) to identify the last character of the user name
    'return the resulting string as the value of the function
    myGetCurrentUser = Left(UserName, InStr(UserName, Chr(0)) - 1)

End Function
```

My thanks to Matthew Dean

Fig 1.5.1 Accessing the Network log-in Id

```
Private Sub Form_Load()

   txtLogin = myGetCurrentUser  'get user login id

'first filter the record for the form using the LogInId (unique primary key)
'filter for matching login id
   DoCmd.ApplyFilter , "[LogInId] = forms![CL Log-on with network password]!txtLogin"

'if a match was found then the LoginID from the table will be in the bound field on the form.
'so this can be checked with what the user has typed in.
   If IsNull(LoginID) Or AccessLevel = 0 Then                'no match for LoginID found
      myDisplayWarningMessage "You are not authorised to use this application"
      DoCmd.Close            'close this form while testing
      'DoCmd.Quit             'to exit Access in final version

   Else                          'valid log in and access level

     'say welcome - not essential, but useful testing aid
      myDisplayInfoMessage "welcome " & Forename & " " & Surname _
                           & vbCrLf & "Your Access Level is " & AccessLevel

      DoCmd.OpenForm "Chelmer Leisure Main Menu"              'open the main menu
      'Visible = False     'remove this line – form hasn't completed loading so can't set it to invisible.
   End If

End Sub
```

The name of my new login form.

Fig 1.5.2 Code for login form with changes shown in bold.

Now, if you open your login form it should say that you are not authorised to use the application, as your network (or home machine) login is not (yet) in the Control Details table.

Add a new row to your Control details table in the password database, and put your network (or home machine) login in the new row. The login form displays then is closed after the 'Welcome' message. If the 'welcome' message was omitted you would be unaware that any log-in check had taken place.

The check could also be coded inside the main menu using DLookup to get at the AccessLevel for the given LoginID, dispensing with the need for a separate login form altogether. See exercise 1.6.6.

1.6 Exercises

1.6.1. Limit the number of tries.

Add a limit (say, three) to the number of times the user can enter incorrect information to the login form of section 1.5. When the limit has been exceeded, close the application.

1.6.2 Allow for case sensitive passwords.

Check the ASCII code of characters for passwords entered into the password form of section 1.5.

The input mask of Password merely means that that the characters entered show as * in the field. Access is not case sensitive, and regards "a" and "A" (for example) as the same thing. However the ASCII character codes (see 'Getting Started' VBA Trainer, Appendix F.1.1) are different, so, by using the Asc function you can find the ASCII code for each character and compare them. The Len and Mid functions will also be useful (see Appendix H.2 of the 'Getting Started' VBA Trainer).

Fig 1.6.1 shows suggested header and logic for a new public function. Put this in a separate Access module, and test it using the Debugger Immediate Window (see 'Getting Started' VBA Trainer section 1.4.3). Then adapt the coding from Fig 1.4.4 to use this new function to compare the user password and the password from the table.

```
Public Function myCheckPassword(prmWord1 As String, prmWord2 As String) As Boolean

Logic:

  Return value = True    (assume words will match)

  If the words are of different length then     (use Len function)
    Return value = False    (unequal lengths, so cannot match)

  Else

    For counter = 1 To length of word         (for each character)
      Convert the next character in each word to ASCII, storing in a variable
      (use Asc and Mid functions)

      If the converted values are different then
        Return value = False    (not the same ASCII code, so words are not equal)
        Make early exit from For loop
      End If

    Next

  End If

End Function
```

See the ‘Getting Started VBA Trainer for further details about the Len, Asc and Mid functions.

Fig 1.6.1 Suggested logic for comparing ASCII values of two words.

1.6.3 Allow user to change their password.

Offer the user the facility to change their password for the login form of section 1.5.

One way is to add a command button to the login form for 'Change Password'.

You will need, of course, to check the existing login and password first.

1.6.4 Password control application.

Create a separate application to allow a user with a specified access level to add/amend data in the Control Details table (new users, change passwords if users have forgotten them, etc).

You will need to check the user’s login and access level in order to restrict access to authorised users.

1.6.5 Combine login form values and network login information.

In the login form of section 1.5, access the network login and close the application if the login entered by the user and the network login are not the same.

This will prevent anyone other than the person logged on to the machine from accessing the application, and is another layer of security.

1.6.6 Put network login check in Main Menu.

As mentioned at the end of section 1.5, put the check for the network login and access level in the Main Menu and store the information in hidden fields on that form.

1.6.7 Use access levels to restrict user's view of data.

Amend your Chelmer Leisure database to restrict what the user can do or see according to the AccessLevel value.

For example: restrict the tabbed forms that can be seen (see 'Getting Started' VBA Trainer Fig 7.2.3); restrict edit permissions; restrict access to some functions such being able to make bookings.

1.6.8 Record who made data changes

When a Membership record is added or edited, and/or when a booking is made, record the login or name of the person who did the transaction plus the date/time.

You will need to add some extra fields to the tables, but these fields need not be shown on the Membership or Booking forms for this exercise (unless you want to do that as well).

1.6.9 Keep a User Log

Create a new table in your Password Control database from Part 1. Some suggested fields are:

- Login and/or name
- Access type (e.g. log-on, log-off)
- System date/time
- Application name (as this database could be used to control to access to several applications).

When your Chelmer Leisure database is opened, add a row to the Password Control database to record log-on details, and when it is closed add a second row to record log-off details.

Some methods of adding a row are:

- Run an Append query .
    - Create the query via the query design window.
    - Add a wizard button to the main menu to generate the code to run the query.
    - Delete the button (the wizard code will not be deleted)
    - Call the new procedure to add a row.
- Use an INSERT embedded SQL statement with DoCmd.RunSQL.
    - See 'Getting Started' VBA Trainer sections 6.3, 6.5, 6.6.
- Use DAO code and INSERT SQL.
    - See sections 3.2.3, 3.2.5 and 3.3.

Possible places to put the code to add a row are the Main Menu Form_Load and Form_Close events.

There is a discussion regarding keeping a user log in chapter 6 of *Robinson*.

PART 2
ERROR TRAPPING AND CUSTOM ERROR MESSAGES

REVIEW OF PART 2

In this part of the document you will see…

- …that errors can be trapped within procedures using On Error. The error number is available in the Err Object. The error description can be referenced by Err.Description. The Resume command is used to resume to the appropriate point to continue.
- …that errors can be trapped at the form or report via the Form_Error or Report_Error event. The error code is available in the DataErr argument. The action to be taken after the error is dealt with is notified to Access by the appropriate value in the Response argument.
- …that a list of some trappable errors is available within the VBA *Help* system.
- …that *Tools→Options, General*, via a code window, has an option to specify that all errors break into code in the Debugger whilst testing.

2.1 Introduction

As you will no doubt have seen when testing your code, Access traps errors and displays error messages. However, not all of the messages may mean much to a user (or sometimes even to a programmer!). It will give your database application a more professional look if you replace these messages by a (kinder) message of your own rather than leaving the user to wonder what the message means, or to intercept some errors and take appropriate action rather than letting the application or process fail (not all errors indicate terminal conditions).

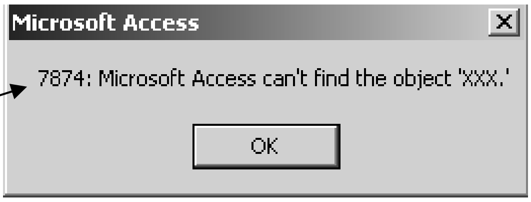
This part of this document will show you how to intercept errors, access the error code and take appropriate action.

You will already have seen some error coding in sections 1.6, 2.5.1 and 2.6.2 of the ‘Getting Started’ VBA Trainer. Wizard code routinely uses error-trapping; Fig 2.1.1 shows wizard code for running a query.

```
Private Sub cmdRunQuery_Click()
On Error GoTo Err_cmdRunQuery_Click

    Dim stDocName As String

    stDocName = "XXX"
    DoCmd.OpenQuery stDocName, acNormal, acEdit

Exit_cmdRunQuery_Click:
    Exit Sub

Err_cmdRunQuery_Click:
    ‘MsgBox Err.Description    ‘Access code
    MsgBox Err & ": " & Err.Description  ‘my code
    Resume Exit_cmdRunQuery_Click

End Sub
```

Query name replaced by one that doesn’t exist, in order to generate error message.

MsgBox statement amended (in bold) to show error code as well (it would be helpful if Access did this each time!)



Microsoft Access

7874: Microsoft Access can't find the object 'XXX.'

Fig 2.1.1 Wizard code to run a query,

The Err Object is used to determine the error code (see VBA *Help* with the keywords *Err Object*). There is more useful information in VBA *Help* using the keywords *Elements of Run-Time Error Handling*, or simply *Error.*

Important: note that the labels Exit_cmdRunQuery_Click and Err_cmdRunQuery_Click both have a colon (:) after them. This shows the compiler that they are labels and not procedure calls. The compiler will give the message 'Sub or Function not defined' if the colon is missing.

Chapter 4 of *Robinson* has a useful discussion of error handling towards the end of the chapter, including a recommendation that all procedures have error-handling code.

2.2 Error Codes

See VBA *Help* with the keywords *trappable errors* (or *error messages)* for a list of codes and errors to which they apply. This is just the list of errors that you can trap within your code; there are other errors that MS Access itself uses (including some that apparently <u>can</u> be trapped by yourself, such as error 7874 seen in Fig 2.1.1).

Two common methods of trapping errors are:

- Coding an On Error clause within a procedure. Code generated by wizards usually uses this method; see Fig 2.1.1 for wizard code to run a query.
- Using an Error event for a form or report. The error code is passed as a parameter.

The next sections show examples of both methods.

Some of the Access messages show what the error code is or provide a *Help* button, and some don't. For the latter type, you will need to do some detective work to work out what the code is, such as searching through the *Help* list mentioned above, using the Debugger or using the Err Object. It can be useful to amend the generated Access use of MsgBox as shown in Fig 2.1.1. Displaying the procedure/form/report name can also be useful in pinpointing just where the error occurred, as demonstrated in Fig 2.3.4.

You may also need some trial and error detective work to find out just where to position the code to trap the error. Use of the Debug facility and breakpoints can be especially useful here. You may also find the following useful:

- Click on *Tools→Options* from a <u>code</u> window
- Choose the *General* tab
- Look at the *Error trapping* section and select *Break on all errors* (the default is to *Unhandled Errors*)

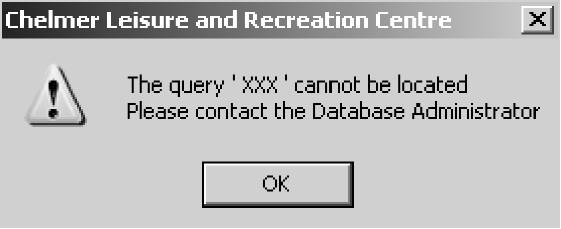
Then you will automatically enter Break mode (i.e. go into the Debugger to look at the code) when an error occurs. This can be useful when testing.

2.3 Coding On Error within a procedure and using the Err Object

2.3.1 Display your own message in place of an Access message

Not all Access messages make sense to a user, so you may want to display your own message instead. For example, the error-handling code in Fig 2.1.1 could be changed to check for error 7874 and display a different message. See Fig 2.3.1; the changes are shown in bold font.

```
Err_cmdRunQuery_Click:
    If Err = 7874 Then        'query is missing or has incorrect name
        myDisplayWarningMessage "The query ' " & stDocName & " ' cannot be located" _
                                          & vbCrLf & "Please contact the Database Administrator"
    Else                      'display Access's own message
        MsgBox Err & ": " & Err.Description
    End If
    Resume Exit_cmdRunQuery_Click

End Sub
```



Chelmer Leisure and Recreation Centre

The query ' XXX ' cannot be located
Please contact the Database Administrator

Fig 2.3.1 Checking for a 'missing query' error and displaying a custom message

This code uses the error number as a literal in the code, but it may be better practice to define all such codes as public constants in an Access module, so that they can be used throughout an application.

```
Code    Public Const myconMissingQuery = 7874        in the module
and     If Err = myconMissingQuery Then              in Fig 2.3.1.
```

Error numbers can change between versions of Access (I have noticed this between Access 97 and Access 2000+), and using constants with meaningful names would make maintenance easier. See exercise 2.5.1.

2.3.2 Suppress an unwanted Access information error message

In section 2.5.2 of the "Getting Started" VBA Trainer, you were shown how to suppress an unwanted 'DoMenuItem cancelled' (Err = 2501) error message caused by cancelling a Save operation. Once you know the error number of the message, suppression is easy by coding something like the code shown in Fig 2.3.1.

Note that you could simply code On Error Resume Next
This would ignore the specific message, but is dangerous as it would also ignore any other messages.

```
Private Sub SubName
On Error GoTo Err_SubName
:
:

Exit_SubName:
   Exit Sub

Err_SubName:
   If Err = 2501 Then            'check for the specific message and ignore it
      'do nothing
   Else
      MsgBox Err & ": " & Err.Description      'for all other errors
   End If
   Resume Exit_SubName
End Sub
```

Fig 2.3.2 Code to suppress Access message for error condition 2501

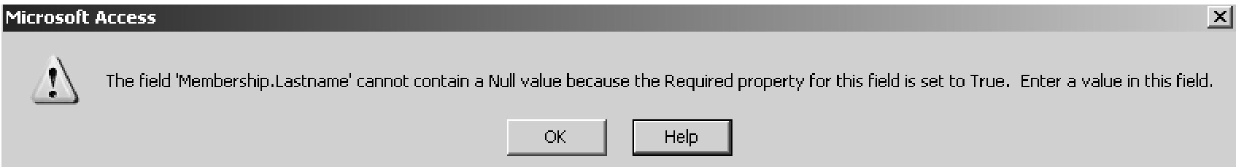
The example shown above is for suppressing an error message, but your own coding for a particular action could be coded here instead if that was more appropriate in a given situation. See Fig 4.3.8 for an example.

2.3.3 Checking for required/mandatory fields

The *McBride* data in Quick Reference 1 specifies some fields that must contain data (Required = Yes); i.e. these are mandatory fields.

One of the *McBride* required fields is the LastName field. Check that you have set Required = Yes for this field in the table, then open your Membership form, and remove the value in this field. When you tab out of the field you will see the message in Fig 2.3.3. (If you have a coded a check for a Null value in the Lastname_BeforeUpdate event then your code will take precedence over the Access Required = Yes check, so comment that out for now, or choose another field to demonstrate this).

If you try to add a new record and miss out some required fields (by tabbing over them for example) then you will get the same message when you attempt to save the record, but (without the useful *Help* button!). Any BeforeUpdate event for the field will not be activated.



The field 'Membership.Lastname' cannot contain a Null value because the Required property for this field is set to True. Enter a value in this field.

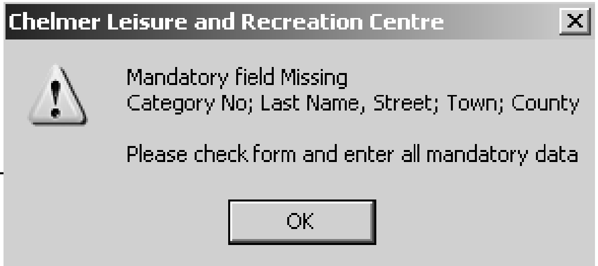
Fig 2.3.3 Standard Access error message when a required field is missing

This is probably one of Access's better error messages (especially the second part), but may still be confusing to a user. If your message has a *Help* button, then this will tell you the error number. Otherwise, display your own message to show this value, or use the Debugger. The code here is 3314.

One way of trapping the situation where the user is attempting to save a new record, and has missed out some mandatory fields is to use the Click event for a Save button. Fig 2.3.4 shows the code for the wizard *Save* command button called cmdSave from Fig 2.5.2 of the 'Getting Started' VBA Trainer. The code has been changed to use Case instead of If.

```
Err_cmdSave_Click:
    Select Case Err

      Case Is = 2501
       'ignore DoMenuItem cancelled message (2501) caused by cancelling changes in BeforeUpdate event

      Case Is = 3314      'mandatory field missing
       'when adding new record and clicking cmdSave - 3314
         myDisplayWarningMessage "Mandatory field Missing" _
                & vbCrLf & "Category No; Last Name, Street; Town; County" _
                & vbCrLf & vbCrLf & "Please check form and enter all mandatory data"
         bSaveError = True

       Case Else
         MsgBox "cmdSave - " & Err & ": " & Err.Description

     End Select

   Resume Exit_cmdSave_Click
```

Sub name added to error message


Chelmer Leisure and Recreation Centre

Mandatory field Missing
Category No; Last Name, Street; Town; County

Please check form and enter all mandatory data

Fig 2.3.4 Checking for missing mandatory fields

Now, if you start a new record and attempt to save it with at least one of the mandatory fields missing, you should get the message shown above and the save will be cancelled. The user must enter the missing value(s) before being able to save the record.

If you use this method, it would also be useful HCI to indicate on the form which fields are mandatory. Many web pages have an asterisk (*) by mandatory fields, so that the user can see what information must be entered.

The Required = Yes condition is also checked when you move to a Next or Previous record, so can be trapped and coded-for. See Exercise 2.5.3.

However, I cannot find out how to trap this situation when the user clicks on a Close button. See the end of section 2.5.2 and exercise 2.7.4 both of the 'Getting Started' VBA Trainer.

Finally note the format of the error message in the Case Else part of the code in Fig 2.3.4. This shows the procedure name where this error message was invoked, which can be useful when Debugging.

2.4 Using the Form_Error Event

This event is invoked automatically when an error occurs on a form. There is a similar event for reports. If this event is missing, the standard Access message is invoked for the error.

2.4.1 Checking a relationship in another table

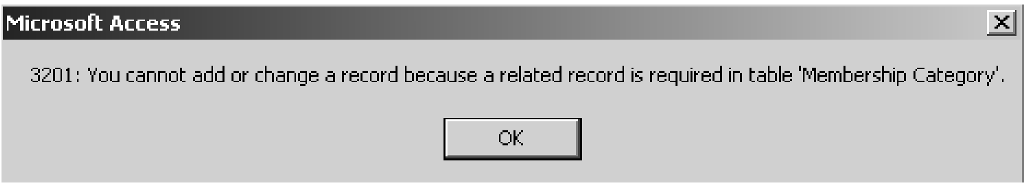
The Membership and Membership Category tables are related via the Category No (Primary Key for the Membership Category table and Foreign Key for the Membership table). Referential integrity should have been enforced, meaning that Access will check that the Category No entered on the Membership form actually matches an entry in the Membership Category table. You may have entered a validation rule that the Category No must be within the range 1-6 (as in *McBride* Quick Reference 1), but a mismatch condition could occur in at least two situations:

- The user has moved the cursor over the Category No field when entering a new member and attempts to save the record without entering anything in here
- The Membership Category table is changed to remove one of the categories, but the validation condition is not changed, or vice-versa (maintenance is a common cause of new errors).

Do the following:

- Change the Membership table/form validation condition for the Category No to be a different (larger) range, or remove the validation rule altogether.

- Change the Category No for a Membership record to 7 (or some other value not in the Membership Category table) and save the record. You should then see a message similar to that shown in Figure 2.4.1 (the error number has been added in the Save_Click event, as seen before).



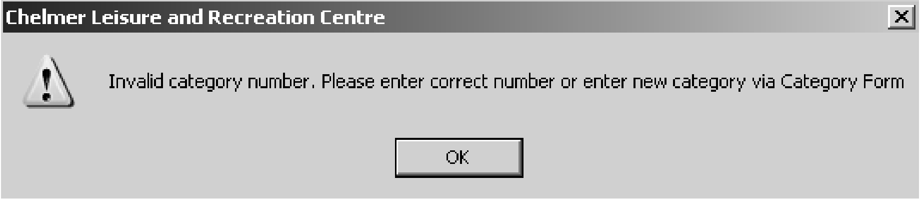
Microsoft Access

3201: You cannot add or change a record because a related record is required in table 'Membership Category'.

Fig 2.4.1 Access error message when related key is not found

This error can be trapped in the Form_Error event. Create this event and add the code shown in Fig 2.4.2.

```
Private Sub Form_Error(DataErr As Integer, Response As Integer)

    If DataErr = 3201 Then        'category number not in category table
      myDisplayWarningMessage "Invalid category number. " & _
              "Please enter correct number or enter new category via Category Form"
      Response = acDataErrContinue   'suppress Access's own message
    Else
      Response = acDataErrDisplay     'Access own message
    End If

End Sub
```



Chelmer Leisure and Recreation Centre

Invalid category number. Please enter correct number or enter new category via Category Form

Fig 2.4.2 Amended code to trap Category No error (top), and the error message (bottom)

Points to note:

- The parameter DataErr is used to tell the procedure the error code for the form error that has just occurred. Use the Debugger, or a simple MsgBox DataErr statement to find out this value. In this example, the value is 3201.
- The Response parameter is for you to tell Access what to do after your code has executed.
  - acDataErrContinue tells Access to continue and will suppress Access's own error message.
  - acDataErrDisplay tells Access to go ahead and display the standard Access error message.
  - You have seen this Response parameter before, in the 'Getting Started' VBA Trainer section 6.3. If you do not return a value, then acDataErrDisplay is assumed.
- The Membership form has only got the one relationship here so it is easy to know which one has caused the error. The error code does not notify which relationship is in error. If there were more than one relationship, the code would need to check the various fields, or the error message would need to spell out clearly which field(s) may be in error and why. In this case, trapping this error here is probably not the best method but has been used just to demonstrate the general principle. In fact, the Category No field is probably best as a LimitToList combo box linked to the Membership Category table.
- Rather than directing the user to enter a new category via the Category form, a much more user-friendly method could be to ask the user if he/she wished to do this and then open the form for them if they replied Yes, provided it was allowable for that user to create new categories, of course! (in which case the user's access level could be checked first; see section 1.4).

2.5 Exercises

2.5.1 Using public constants for error codes

As suggested in section 2.1.1, create Public Constants for all the error codes so far, and replace the literals in the code with references to the relevant constant.

2.5.2 Own messages for start/end of a set of records on a form

On a form which has buttons to move to the Next and Previous records, check the error codes for the messages that display when you reach the start/end of the set of records and display messages of your own in place of the Access messages.

2.5.3 Trapping missing fields when moving to next/previous records

Add wizard *Next* and *Previous* buttons to your form (if they are not there already).

Add a new record, but leave at least one mandatory field missing. Move to the next/previous record and see the error caused. Work out the error code for the error and then trap the error, and display a ‘mandatory fields missing’ message.

If you have a ‘Save Changes’ procedure, you may need to use a flag to suppress the message if the user replies No and the changes are cancelled. Use *Help* to see the various Resume options that are available.

PART 3 – USING DATA ACCESS OBJECTS (DAOs)

REVIEW OF PART 3

In this part of the document you will see…
- …definitions of DAO and ADO.
- …various useful *Help* references and other information sources
- …how to install the DAO library in Access 2000+.
- …how to declare a Database Object and assign it to the current or an external database.
- …how to use the Database Execute method to perform actions on a table.
- …how to declare and populate a Recordset Object, and use various methods and properties to read from and write to a table in the current or an external database.
- …how to add to a combo list at run-time using DAO code.
- …how to create your own Domain Aggregate functions using DAO code.
- …how to use the Object Browser to see more about DAO Objects.

3.1 Introduction

3.1.1 Data Access Objects (DAO).

- “A data access interface that communicates with Microsoft Jet and ODBC-compliant data sources to connect to, retrieve, manipulate, and update data and the database structure”.
  Access/VBA 2002 Help
- "The container for all the objects that can be embodied in an Access application, often abbreviated *DAO*. The top member of the data access object hierarchy of access is the DBEngine object, which contains Workspace, User and Group objects in collections. Database objects are contained in Workspace objects".
  Jennings R, 1997, *Using Access 97, the Ultimate Reference*, QUE, p 1273.
- "The Microsoft Data Access Objects (DAO) library contains the objects, methods and constants you use to work directly with database files".
  Callahan E, 1995, *Microsoft Access/Visual Basic [Step by Step]*,p 263.

So far you have seen how to use queries, forms and reports, and write code to manipulate the data, cater for events, run queries, use embedded SQL, use Domain Aggregate functions, etc. By using DAOs you can get far closer to objects in the database. DAOs were the standard method of doing this prior to Access 2000+.

DAO code is essential when what you want to do is too complex to be achieved by the other facilities available in Access, but it is simple to use. It is also fun to use if you like programming!

3.1.2 ActiveX Data Objects (ADO).

From Access 2000, ADOs became the standard (and default) method of accessing data. However, all DAO code is still included and (according to what I have seen in VBA *Help*) seems to be used in some ADO processes. ADO code seems to be more complex than DAO code but may allow more flexibility and a wider range of functions. (But having acronyms of DAO and ADO is very confusing!)

Some definitions, all from Access/VBA 2002 *Help*:
- “ActiveX Data Objects (ADO) enable you to manipulate the structure of your database and the data it contains from Visual Basic. Many ADO objects correspond to objects that you see in your database — for example, a Table object corresponds to a Microsoft Access table. A Field object corresponds to a field in a table”.
- “Microsoft Access includes ActiveX Data Objects (ADO) 2.5 as the default data access library. Although Data Access Objects (DAO) 3.6 is included it is not referenced by default”.
- “Microsoft ActiveX Data Objects (ADO) provides the objects, such as tables, queries, relationships, and indexes, that handle data-management tasks in a Microsoft Access database. These objects are called data access objects. You can share Visual Basic code that uses data access objects with other applications that use Microsoft ADO, such as Microsoft Excel”.

I have not had time to explore ADO code, so this Part of the Trainer deals only with DAO code. However, if you know how one works, you should (she says…) be able to understand the other. If you use the keyword *ActiveX* in VBA *Help*, then select *Microsoft ADO Programmer's Reference* you will see a link to Getting Started with ADO.

Another VBA *Help* reference that you might find useful is to be found via the keyword *Convert*, then select the item *Converting DAO Code to ADO.*

3.1.3 Some references for further information.

Access 2000/2002 Help:

- VBA help - Index keywords:
    - *Database; object* for items on this topic
    - *object; browser* for details of the Object Browser
- VBA Help - Answer wizard:
    - *DAO* for list of Recordset methods and properties
    - *Converting DAO code to ADO code* for how to convert code

Access 2003 Help:

- VBA Help – search box
    - *Recordset* for information about Recordsets
- VBA help – Table of contents
    - *Microsoft DAO 3.60* for information about DAOs
- Object Browser Help – search box (see also section 3.5)
    - *Object Browser* in Search box in Object Browser Help
    - *Database Object* for items on this topic
    - *DAO* for information about DAOs
    - *Convert code* for information about converting DAO code to ADO

There are some example databases on http://www.cse.dmu.ac.uk/~mcspence/Access.htm that demonstrate DAO code.

Section 4 of this document has a large example of using DAO code and arrays to create a Booking Table diary page grid.

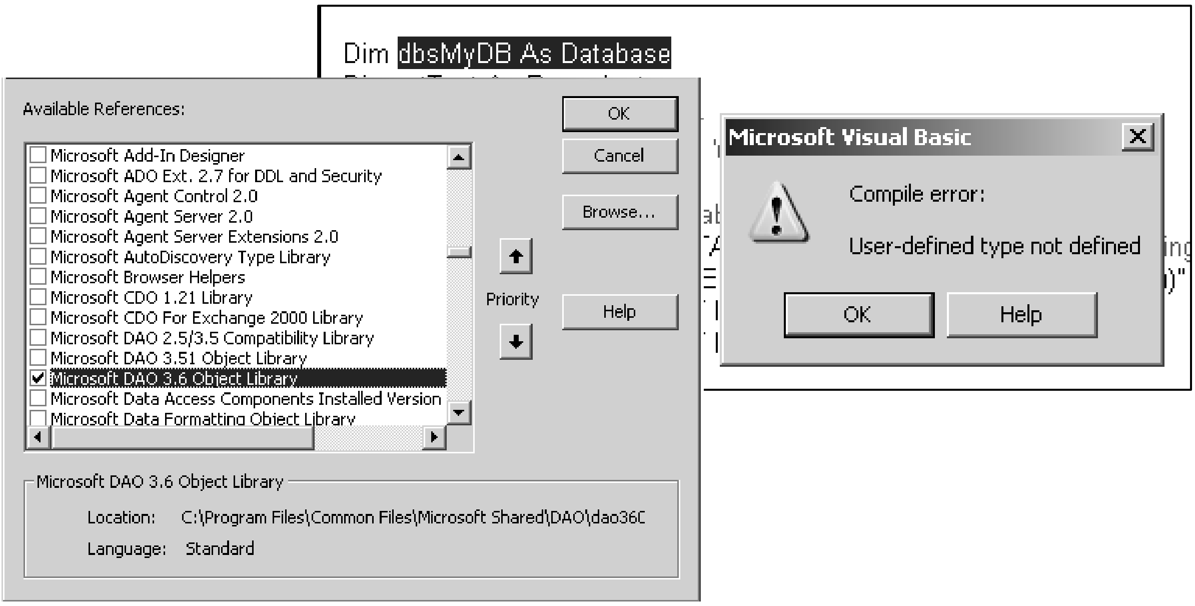
3.1.4 Using DAO code in Access 2000+

As stated in 3.1.2, ADOs are the standard method expected by Access, and the default libraries have been installed to expect ADO code. The **DAO Object Library** therefore needs to be installed in order to code in Access 2000+ using DAOs.

To install the DAO Object Library, do:

1. Open any code window
2. Choose *Tools→References*
   (note that the *Tools* menu gives a different list to the Access one, for an open code window).
3. Click the checkbox for Microsoft DAO Library
   (probably version 3.6 for Access 2000/2002/2003).

Fig 3.1.1 shows the Dialog box resulting from *Tools→References* and a compilation error that will occur if you attempt to define a variable of type Database without first installing the DAO Object Library.



```
Dim dbsMyDB As Database
```

Available References:

☐ Microsoft Add-In Designer
☐ Microsoft ADO Ext. 2.7 for DDL and Security
☐ Microsoft Agent Control 2.0
☐ Microsoft Agent Server 2.0
☐ Microsoft Agent Server Extensions 2.0
☐ Microsoft AutoDiscovery Type Library
☐ Microsoft Browser Helpers
☐ Microsoft CDO 1.21 Library
☐ Microsoft CDO For Exchange 2000 Library
☐ Microsoft DAO 2.5/3.5 Compatibility Library
☐ Microsoft DAO 3.51 Object Library
☑ Microsoft DAO 3.6 Object Library
☐ Microsoft Data Access Components Installed Version
☐ Microsoft Data Formatting Object Library

Microsoft DAO 3.6 Object Library
Location: C:\Program Files\Common Files\Microsoft Shared\DAO\dao36C
Language: Standard

Microsoft Visual Basic

Compile error:

User-defined type not defined

Fig 3.1.1 How to install the DAO object library, and the error message if the library is not installed.

If you move the DAO library reference to above the ADO library reference as shown in Fig 3.1.2, then you can continue to code using DAO code as in earlier versions of Access. If the ADO library reference comes first, then that will take precedence, and you need to prefix code with *DAO* or you will get a *Type Mismatch* error. In this case you should code each Recordset declaration as

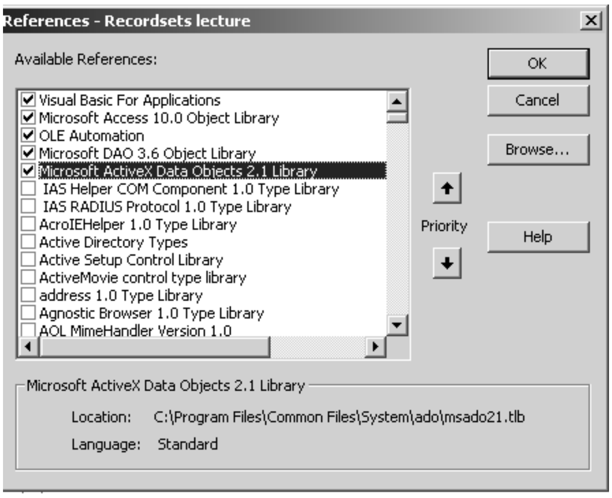
```
        Dim rstRecords As DAO.Recordset
```

in place of

```
        Dim rstRecords As Recordset
```

See also the *VBA and Access FAQ* at http://www.cse.dmu.ac.uk/~mcspence/Access.htm

It may be safest to use the *DAO* prefix regardless of the order of the libraries.



References - Recordsets lecture

Available References:

☑ Visual Basic For Applications
☑ Microsoft Access 10.0 Object Library
☑ OLE Automation
☑ Microsoft DAO 3.6 Object Library
☑ Microsoft ActiveX Data Objects 2.1 Library
☐ IAS Helper COM Component 1.0 Type Library
☐ IAS RADIUS Protocol 1.0 Type Library
☐ AcroIEHelper 1.0 Type Library
☐ Active Directory Types
☐ Active Setup Control Library
☐ ActiveMovie control type library
☐ address 1.0 Type Library
☐ Agnostic Browser 1.0 Type Library
☐ AOL MimeHandler Version 1.0

Microsoft ActiveX Data Objects 2.1 Library
Location: C:\Program Files\Common Files\System\ado\msado21.tlb
Language: Standard

Fig 3.1.2 Showing DAO library installed before ADO library.

3.2 Basics of DAO code

3.2.1 Defining a Database Object

A Database Object represents an open database.

In order to reference a DAO object within VBA you first need to declare a Database Object variable:

```
Dim dbsTheDatabase as Database
```

See also Fig 3.1.1.

Then you need to tell Access which database you are using:

```
Set dbsTheDatabase = CurrentDb                                  'the database in which this code resides
Set dbsTheDatabase = DBEngine.Workspaces(0).OpenDatabase(strPath)  'an external database
```

See also Fig 3.2.1. The Set statement is an assignment statement.

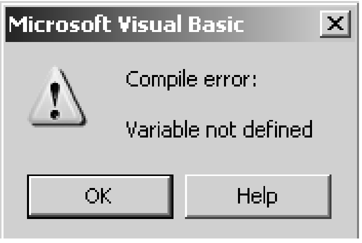
The code here shows how to define a Database Object, then to say which database is to be referenced. You need the Dim statement first then the Set statement.

You can access objects within the current database and/or an external database. For an external database you need to specify the pathname as a string; the code above assumes that this is in the variable (or it could be a constant) strPath

If you position the cursor on CurrentDB and hit the F1 key, you will see VBA *Help* which includes the following:

- "In Microsoft Access the CurrentDb method establishes a hidden reference to the Microsoft DAO 3.6 Object Library in a Microsoft Access database (.mdb)".
- "In order to manipulate the structure of your database and its data from Visual Basic, you must use Data Access Objects (DAO). The CurrentDb method provides a way to access the current database from Visual Basic code without having to know the name of the database. Once you have a variable that points to the current database, you can also access and manipulate other objects and collections in the DAO hierarchy".

If you omit the Dim statement for the Database Object, then you will get a compile error as shown in Fig 3.2.1 for the first Set statement, just as you would when assigning any value to an undefined variable.



Compile error:

Variable not defined

Fig 3.2.1 Error if attempting to Set a database value without first defining the database

3.2.2 The Execute method

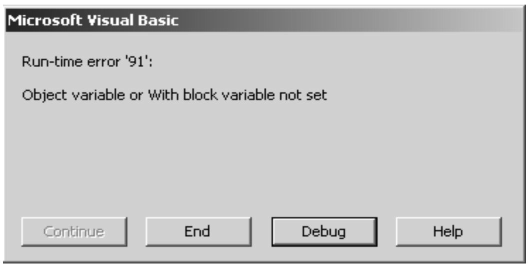
A useful method of the Database Object is the Execute method. This can be used to run SQL for action queries and is therefore an alternative to using DoCmd.RunSQL. The SQL could also be put into a string variable first, just as with the RunSQL method of DoCmd.

The examples below show how to create, add a row to, and delete, a simple table called TestDAO, which has just one field called TestData.

```
dbsTheDatabase.Execute "CREATE TABLE TestDAO (TestData CHAR(10))“
dbsTheDatabase.Execute "INSERT INTO TestDAO VALUES ('NewData')“
dbsTheDatabase.Execute "DROP TABLE TestDAO“
```

There are many other Database methods available; see VBA *Help* or the Object Browser (section 3.5) for further details. You will see the OpenRecordset method in section 3.2.4.

If you have omitted to run a Set statement for the Database Object before a statement that refers to that database object, then you will get the run-time error shown in Fig 3.2.2. The bit of the message that applies here is 'object variable not set'.



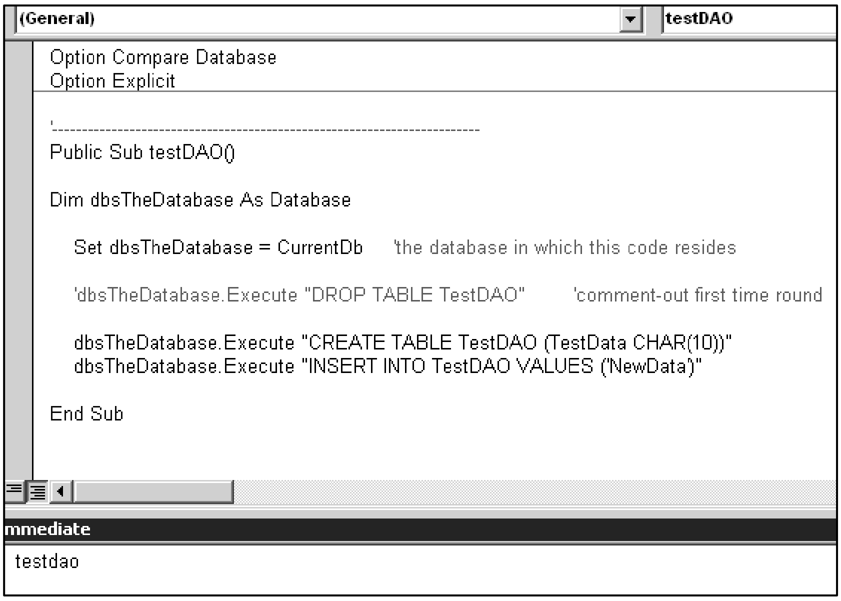
Run-time error '91':

Object variable or With block variable not set

Fig 3.2.2 Error if omit to run the Set statement before referencing a Database Object

3.2.3 A very simple example to start with

In your Chelmer Leisure database, create a new module with some DAO code, as shown in Fig 3.2.3. This uses some of the code from the preceding sections.



```
Option Compare Database
Option Explicit

'------------------------------------------------------------------------
Public Sub testDAO()

Dim dbsTheDatabase As Database

    Set dbsTheDatabase = CurrentDb     'the database in which this code resides

    'dbsTheDatabase.Execute "DROP TABLE TestDAO"        'comment-out first time round

    dbsTheDatabase.Execute "CREATE TABLE TestDAO (TestData CHAR(10))"
    dbsTheDatabase.Execute "INSERT INTO TestDAO VALUES ('NewData')"

End Sub
```

```
testdao
```

Fig 3.2.3 Simple procedure to create a table and add a row to it.

Run the procedure in the Debugger by simply typing the name in the Immediate Window (it is a Public sub; if it were defined as Private you would get the error *Sub or Function not defined* as the Debugger cannot access it) and then pressing the Enter key.

Note that the line to DROP (delete) the table is commented out, as the table does not exist yet, so cannot be deleted.

Now look at the list of tables in the database window, and you should see the new table there with the single row of data, as shown in Fig 3.2.4.

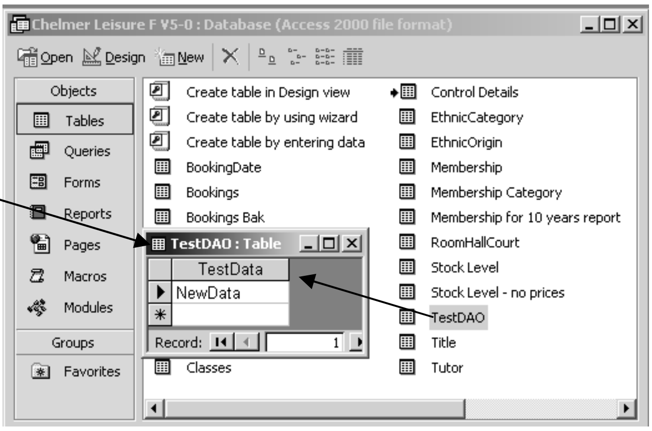
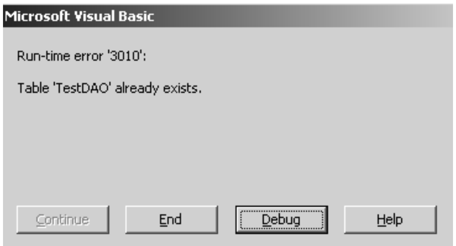


Fig 3.2.4 database window showing the new table and the table contents.

Now remove the comment apostrophe from the line of code that DROPs the table (so that this code is no longer a comment), change the VALUES in the INSERT statement, and run the procedure again. (If you attempt to run the code again without the DROP statement you will get the error shown in Fig 3.2.5 as you are attempting to CREATE a table that already exists).


Microsoft Visual Basic

Run-time error '3010':

Table 'TestDAO' already exists.

Fig 3.2.5 Error message if attempt to create a table that already exists.

Look at the table in your database, and see that the row of data has changed. The original row has gone, and is replaced by the row from the second run of the code.

3.2.4 Recordset Object, methods and properties

“All Recordset Objects consist of records (rows) and fields (columns)”. *VBA 2002 Help.*

"Recordset objects … represent virtual tables (images) that are stored in RAM. Recordset objects are said to be 'created over' a table or a query result set". *Jennings R, 1997, Using Access 97, the Ultimate Reference, QUE, p 1046.*

So far, we have manipulated data by methods such as…

- …running queries on them from the database window or via wizard code
- …using Domain Aggregate functions
- …using embedded SQL in code to run action queries using DoCmd.RunSQL or Execute.

With Recordsets you can ‘get your hands on’ the data records and manipulate them more closely. You are more in control of what is going on.

Just as for a Database Object, the first thing you must do is to declare a Recordset Object variable. See also section 3.1.4.

```
        Dim rstTest As Recordset
```

Access 97 declaration; OK with 2000+ if DAO library takes precedence over DAO.

```
        Dim rstTest As DAO.Recordset
```

Statement using DAO prefix, to force use of DAO library.

Fig 3.2.6 lists some useful Recordset methods and properties, some of which will be demonstrated on the next few pages. Look at VBA *Help* for the full list.

Type	Name	Description
Some useful Methods	AddNew	Add a new record (row) to a table
	Delete	Delete a row from a table
	Edit	Change a row in a table
	Close	Close a Recordset (as it is locked while open)
	MoveFirst/Last/Previous/Next	Move to records within a Recordset
	Requery	Populate the Recordset again and reset the RecordCount
	Seek	Search for a specific record by key
	Update	Complete all pending updates for the row
Some useful Properties	BOF, EOF	Beginning/End Of File (Boolean value. TRUE if BOF/EOF reached; FALSE otherwise)
	Bookmark	Identifies the required record (used for wizard filters)
	Fields	Use to read and to set values for the fields in the current record of the Recordset object.
	NoMatch	Used with Seek to indicate if the key was found
	RecordCount	Gives the number of records accessed in the Recordset

Fig 3.2.6 Some useful Recordset Methods and Properties

Once the Recordset Object has been declared you can assign a value to it. Again, just like the Database Object you use a Set statement:

```
  Set rstTest = dbsTheDatabase.OpenRecordset("SELECT * FROM TestDAO")  'select everything
  Set rstTest = dbsTheDatabase.OpenRecordset("TestDAO")                 'alternative method to select everything
  Set rstTest = dbsTheDatabase.OpenRecordset(strSQL)                    'SQL in a String variable
‘select specified field(s) with a WHERE clause
  Set rstTest = dbsTheDatabase.OpenRecordset("SELECT TestData FROM TestDAO WHERE TestData = ‘NewData’ ")
  Set rstTest = dbsTheDatabase.OpenRecordset(qryTest)                   ‘using a query – can select all or just some
```

These statements reference the database variable dbsTheDatabase and use the OpenRecordset method of the Database Object to create a Recordset which contains the required data from the table TestDAO (i.e. to populate the Recordset with data). The Recordset is thus effectively a copy of the table/query put into memory for code to access.

3.2.5 A more complex example

This example puts the various topics in the previous sections together to…

- …declare and assign Database variables for the current database and an external database
- …declare and assign Recordsets for each of the two databases
- …create and populate a table in the current database
- …read the records from this table and write them to a table in the external database.

This example may not be of any immediate practical use, but is used here to illustrate how to code the elements and use some Recordset methods.

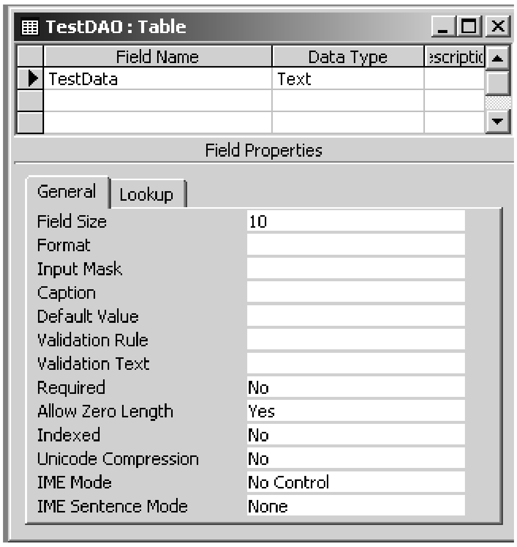
An external database can be useful for:

- Recording who has used the database, by writing login names (See Part 1), date and time on/off.
- Recording changes made to a database since the last back-up. This can then be used as part of the recovery procedure in case of data loss.

This example uses the databases and tables shown in Fig 3.2.7.

- Table TestDAO is the table from section 3.2.3. This table is in your Chelmer Leisure database, so is in the current database.
- For the other (external) table, create a new database, and add a table with one field, all with the names shown.

Database		**Table**			**Location**
Name	**Database variable**	**Name**	**Field**	**Recordset variable**	
CL database used for this Trainer	dbsMyDB	TestDAO	TestData	rstTest	Current
OtherDBTest.mdb	dbsOtherDB	TestTable	TestField	rstOther	External



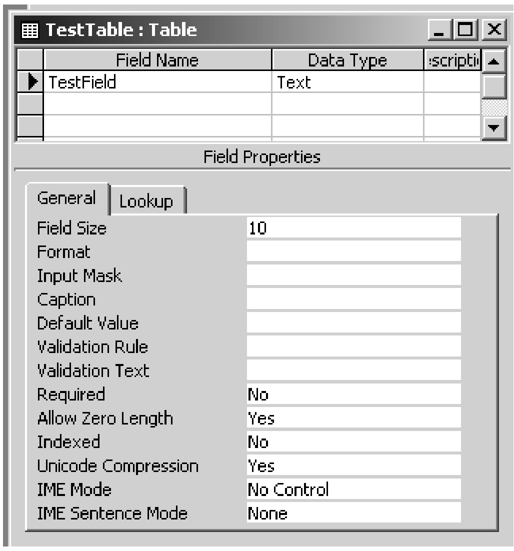
TestDAO : Table

Field Name	Data Type	Descriptio
TestData	Text	

Field Properties

General | Lookup

Field Size	10
Format	
Input Mask	
Caption	
Default Value	
Validation Rule	
Validation Text	
Required	No
Allow Zero Length	Yes
Indexed	No
Unicode Compression	No
IME Mode	No Control
IME Sentence Mode	None



TestTable : Table

Field Name	Data Type	Descriptio
TestField	Text	

Field Properties

General | Lookup

Field Size	10
Format	
Input Mask	
Caption	
Default Value	
Validation Rule	
Validation Text	
Required	No
Allow Zero Length	Yes
Indexed	No
Unicode Compression	Yes
IME Mode	No Control
IME Sentence Mode	None

Fig 3.2.7 The databases and tables used in this example.

The code is shown in Fig 3.2.8, with some explanation on the following page. The comments are shown in italics so that the code can be seen more clearly.

Use the Debugger to test the code, check the values in variables and properties, and see how it all works.

```
Public Sub TestDAO2()

'declare variables for the two databases
Dim dbsMyDB As Database           'for the current database
Dim dbsOtherDB As Database        'for the external database

'declare Recordset variables for the two tables
Dim rstTest As DAO.Recordset      'current
Dim rstOther As DAO.Recordset     'external

'set up a constant for the path for the external database
Const OtherDBFullPathName = "C:\Documents and Settings\Mary\My Documents" _
                              & "\VBA stuff\Trainer v5\FVBA v5-0\OtherDBTest.mdb"
'A path for the current database is not needed, as this code knows where that is

'assign the current database to the appropriate variable
  Set dbsMyDB = CurrentDb

'assign the external database to the appropriate variable
 'Using a path relative to 'My Documents'
  'Set dbsOtherDB = DBEngine.Workspaces(0).OpenDatabase("VBA stuff\Trainer v5\FVBA v5-0\OtherDBTest.mdb")
 'Using the full path reference, including drive letter (can reference other drives this way)
  Set dbsOtherDB = DBEngine.Workspaces(0).OpenDatabase(OtherDBFullPathName)

'Have now declared variables for the two databases and assigned values to the variables.

'create and populate a test table in the current database
  dbsMyDB.Execute "DROP TABLE TestDAO"  'so can re-run code"
  dbsMyDB.Execute "CREATE TABLE TestDAO (TestData CHAR(10))"
  dbsMyDB.Execute "INSERT INTO TestDAO VALUES ('NewData')"
  dbsMyDB.Execute "INSERT INTO TestDAO VALUES ('MoreData')"

'open a Recordset for the test table records in this database
  Set rstTest = dbsMyDB.OpenRecordset("SELECT * FROM TestDAO")

'open a Recordset for the Other database
  Set rstOther = dbsOtherDB.OpenRecordset("TestTable")

'start to process the data
  rstTest.MoveLast        'this will update the RecordCount
  MsgBox rstTest.RecordCount & " records in table"   'display count

  rstTest.MoveFirst         'move back to first record
  rstTest.Requery           'reset RecordCount and repopulate Recordset

'process each record in the Recordset
  'this reads each record from the table in the current database
  'and writes data to the table in the external database
   Do Until rstTest.EOF
     MsgBox rstTest.RecordCount & " " & rstTest.Fields("TestData")
     rstOther.AddNew                     'specify action to be taken
     rstOther!TestField = rstTest!TestData  'details of the action
     rstOther.Update                     'apply the specified action
     rstTest.MoveNext      'get next record in Recordset
   Loop

   MsgBox "All records read"   'testing only

   'can get Error 3211 if  forget these lines - 'table already in use by another person or process'
   rstTest.Close
   rstOther.Close
   dbsMyDB.Close
   dbsOtherDB.Close

   Set rstTest = Nothing   'disassociate Recordset from the table

End Sub
```

Declare Database and Recordset variables

Assigning values to the database variables. Two methods are shown for the external database.

Code already seen in section 3.2.3, to add rows to a temporary table.

Opening the two Recordsets with the required data. Two different methods of reading the full data are shown.

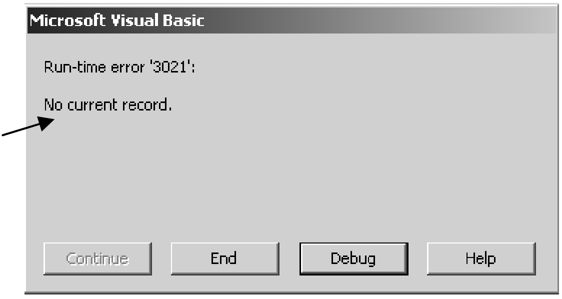
Two different ways of referencing a field in a Recordset.

See explanation on next page.

Fig 3.2.8 Example showing how to use DAO code with the current and an external database, reading records from one table and writing to another.
The path for the external database is for my table on my machine. You will need to change this to match the path for your machine.

Explanation of code in Fig 3.2.8.

- *The code starts by declaring the variables for the two Databases and the two Recordsets, as seen in sections 3.2.1 and 3.2.4.*
- **Const OtherDBFullPathName = "C:\Documents and Settings\Mary\My Documents" _**
  & "\VBA stuff\Trainer v5\FVBA v5-0\OtherDBTest.mdb"
  - o *You need to specify the path for the external database. This shows how to set up a constant for the path. Note that this code uses the full path and drive (of my file on my machine).*
- **Set dbsMyDB = CurrentDb**
  'Set dbsOtherDB = DBEngine.Workspaces(0).OpenDatabase("VBA Stuff\OtherDB Test.mdb")
  Set dbsOtherDB = DBEngine.Workspaces(0).OpenDatabase(OtherDBFullPathName)
  - o *Assign specific databases to the Database Objects. You can use the full pathname for the external database (and must do this if the database is on another drive), or the path relative to 'My Documents' if the database in on your own machine, as it may be when testing).*
- *The next lines delete, create and populate the TestDAO table, as already seen in section 3.2.3.*
- **Set rstTest = dbsMyDB.OpenRecordset("SELECT * FROM TestDAO")**
  Set rstOther = dbsOtherDB.OpenRecordset("TestTable")
  - o *These statements read the data from the tables into the Recordsets, as seen in section 3.2.4.*
  - o *Two different methods are shown here; they both read the full data.*
- **rstTest.MoveLast**
  MsgBox rstTest.RecordCount & " records in table"
  - o *These two lines are here merely to demonstrate the* MoveLast *method.*
  - o *When a Recordset is first opened, the first record (if any) is ready for accessing (it is the 'current record').*
    - ▪ *The* RecordCount *property = 0 if there are no records in the Recordset and 1 if there is at least one record.*
  - o *The* RecordCount *property holds the count of records read so far.*
  - o *The* MoveLast *method will read through all the records to the last one. The* RecordCount *will now be equal to the number of records in the Recordset. If the Recordset is empty (i.e. there are no records in it) the* MoveLast *method will fail (as will a* MoveFirst *or* MoveNext *statement). See Fig 3.2.9.*
  - o *The* MsgBox *statement is just for testing.*
  - o *This method may have a negative affect on performance (not efficient).*


Microsoft Visual Basic

Run-time error '3021':

No current record.

Fig 3.2.9 Error if attempt to access a record that does not exist

- **rstTest.MoveFirst**
  rstTest.Requery
  - o *The* MoveFirst *method will position back at the start of the Recordset, so that the first record is the current record. But this will not reset the* RecordCount*. It will fail if there are no records; see Fig 3.2.9.*
  - o *The* Requery *method will requery (and thus repopulate) the Recordset and will also reset the* RecordCount *(so the first line is not actually necessary). Note that you can change the query for the* OpenRecordset *method and the* Requery *method would then read in the records based on the new query (just like using* Requery *on combo and list boxes).*
- **Do Until rstTest.EOF**

```
    MsgBox rstTest.RecordCount & " " & rstTest.Fields("TestData")
    rstOther.AddNew
    rstOther!TestField = rstTest!TestData
    rstOther.Update
    rstTest.MoveNext
  Loop
```

  - o *This a simple loop that reads each record one by one until end of file (*EOF*).* EOF *property =* True *when* EOF *has been reached,* False *otherwise.*
    - ▪ *The* MsgBox *command is just for testing. The* Fields *property is used to reference the field in the current record. The field name is a string, enclosed in brackets.*
    - ▪ *This is the code that updates the external table, and adds the new entry.*
      - ▪ *First the* AddNew *method is used to tell Access that preparations are being made to add a new row.*

    - *Then the row is set up. This row only has one field, but if there were several fields in the row, then you would code a line for each field. See example in Fig 4.3.8.*
      - *Note that this uses a different method from that on the* MsgBox *statement. Either method of referencing a Recordset field is OK.*
    - *No changes are made to the table until the* Update *method has been executed.*
  - *The* MoveNext *method moves to the next record in the Recordset and increments the* RecordCount *by one. This statement must be executed in the loop, or else the loop will continue indefinitely, just processing the same record each time.*

- **rstTest.Close**
rstOther.Close
dbsMyDB.Close
dbsOtherDB.Close
  - *These statements tell Access that you've now finished with the Recordsets and the Databases, and will release the locks. You may get error 3211 if you forget to do this (see sections 3.6 and 4.3.5)*
- **Set rstTest = Nothing**
  - *Not essential in this example, but here as an example of using the* Nothing *keyword to clear the Recordset and to disassociate it from the table, so that you are aware that this keyword exists. This seems to be an alternative to using* Close*.*
  - *If you look in the Debugger, you will see that the Recordset =* Nothing *initially.*

3.3 Adding to a combo box list at run-time

In section 6.3 of the 'Getting Started' VBA Trainer you saw how to use embedded SQL using DoCmd.RunSQL to update a form combo box based on a table. The code in Fig 3.3.1 shows how to do exactly the same thing but using DAO code.

```
Private Sub Title_NotInList(NewData As String, Response As Integer)
'FVBA - using DAO code

Dim strTitleQuestion As String          'question to user
Dim dbsWhichDB As Database              'refers to the required database
Dim rstTitleRecords As DAO.Recordset        'represents records in the required table

strTitleQuestion = " ' " & NewData & " ' is not in the list." _
                   & vbCrLf _
                   & "Do you want to add this title to the list?"

  If myYesNoQuestion(strTitleQuestion) = vbNo Then
    Response = acDataErrContinue
    ' Do nothing. User will automatically be put back to the list.
  Else
    Set dbsWhichDB = CurrentDb()                              'tell Access to use current database
    Set rstTitleRecords = dbsWhichDB.OpenRecordset("Title")    'open Title table
     With rstTitleRecords
       .AddNew                  'specify Add action
       .Fields("Title") = NewData   'set the title field to the new value
       .Update                   'save the change
     End With
    'tell Access what you've done - the combo box list will be requeried
    Response = acDataErrAdded
    rstTitleRecords.Close
    dbsWhichDB.Close
  End If

End Sub
```

Fig 3.3.1 DAO code for NotInList event to add entries to Title table via combo box

Explanation of the code in Fig 3.3.1:

- **Dim strTitleQuestion As String**
Dim dbsWhichDB As Database
Dim rstTitleRecords As DAO.Recordset
  - *Declaring a String variable for a question, a Database variable for the database and a Recordset variable for the table.*

- **strTitleQuestion = " ' " & NewData & " ' is not in the list." _**
  & vbCrLf _
  & "Do you want to add this title to the list?"
  - *Setting-up the question.*
- **If myYesNoQuestion(strTitleQuestion) = vbNo Then**
  Response = acDataErrContinue
  - *If user says No, then tell Access to carry on and refuse to accept the new value. The standard Access error message will not be displayed.*
- **Else**
  Set dbsWhichDB = CurrentDb()
  Set rstTitleRecords = dbsWhichDB.OpenRecordset("Title")
  With rstTitleRecords
  .AddNew
  .Fields("Title") = NewData
  .Update
  End With
  Response = acDataErrAdded
  rstTitleRecords.Close
  dbsWhichDB.Close
  - *This is the bit that updates the table, and adds the new entry.*
    - *Just as for the example in Fig 3.2.8, the code first specifies the type of action (Add, in this case) to be taken, then prepares the row to be added, then tells Access to go ahead and update the table. See section 4.3.5 for an example of code that adds a record with several fields.*
    - *This code demonstrates that you can use the With statement (see 'Getting Started' VBA Trainer section 7.7) with DAO code.*
- **End If**
  - *This just ends the If block.*

3.4 Creating your own domain aggregate functions

In the 'Getting Started' VBA Trainer you saw how to use the Access built-in Domain Aggregate functions DLookup, DCount, etc. This section shows how you can use DAO code to create your own versions (which I shall refer to as the myD functions) of the VBA Domain Aggregate functions, so serves as further examples of DAO code and may also help you understand better what the Access Domain Aggregate functions are doing.

First create an unbound form with two non-wizard buttons, as in Fig 3.4.1.

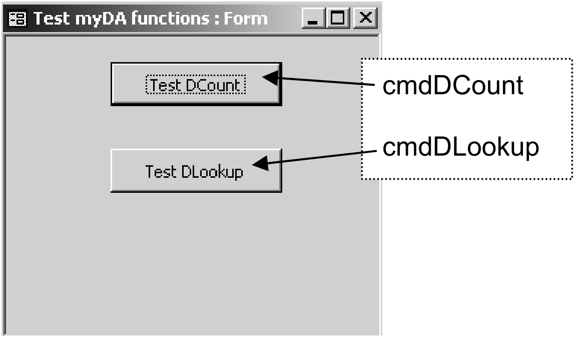


Fig 3.4.1 Form used to test the myD functions

3.4.1 myDCount function – version 1

The Access DCount function has the basic format:**DCount(expression, domain, [criteria])**
Where:

- **Expression** = the name of the table field that you are looking for. The SELECT part of the SQL statement. *Datatype = String.*
- **Domain** = the name of the table or query. The FROM part of the SQL statement. *Datatype = String.*
- **Criteria** = the criteria that you wish to apply. The WHERE part of the SQL statement, with exactly the same format. This parameter is optional. *Datatype = Integer.*

You have seen an example of this function in section 3.4.1.2 of the 'Getting Started' VBA Trainer.

Create a new Access code module and add the code shown in Fig 3.4.2, for the public function myDCount. This uses DAO code to do the same task as the Access DCount function. Note that the function header (apart from the name) is the same as that required for the Access DCount function.

```
Public Function myDCount(prmExpr As String, prmDomain As String, Optional prmCriteria As String) As Integer

Dim myDB As Database              'declare Database variable
Dim rsData As DAO.Recordset       'declare Recordset variable
Dim strSQL As String              'used for the SQL for the data

  Set myDB = CurrentDb        'specify database = assume the current database
..................................................................................
  'create SQL to select count of all records
  strSQL = "SELECT " & prmExpr & " FROM " & prmDomain

  'if a criterion is present, then add this to the SQL
  If IsMissing(prmCriteria) Or prmCriteria = "" Then   'will also be True if prmCriteria empty
     'do nothing
  Else
     strSQL = strSQL & " WHERE " & prmCriteria      ‘add the WHERE clause
  End If

  Set rsData = myDB.OpenRecordset(strSQL)   'get records using given SQL
..................................................................................
  If rsData.RecordCount = 0 Then   'no rows found - RecordCount = 0
     'do nothing                   ‘see Fig 3.2.9
  Else
     rsData.MoveLast               ‘move to last record to update RecordCount
  End If

  myDCount = rsData.RecordCount    ‘return count of records
..................................................................................
  rsData.Close      'close the Database...
  myDB.Close        '... and the Recordset

End Function
```

1. Declare variables

2. Set up the SQL to read records for the Recordset. The WHERE clause is added if the optional parameter is present.

3. Determine number of records in the Recordset and return this value

4. Close the Database and Recordset

Fig 3.4.2 myDCount function version 1

The code above has four elements, divided by dotted lines. You should be familiar with all the code here:

1. Declares the necessary variables and specifies the database to be used.
2. Creates the SQL to select the required rows. If the optional parameter prmCriteria is not present or is empty then there is no WHERE clause. If it the optional parameter is present then the WHERE clause is added. The Recordset can now be opened using this SQL.
3. If the RecordCount property = 0 then the Recordset is empty, so there is nothing further to be done (and MoveLast etc will fail; see Fig 3.2.9). Otherwise, the MoveLast method is used to read to the last record and update the RecordCount. The RecordCount can now be returned as the function value.
4. Finally, the Database and Recordset are closed.

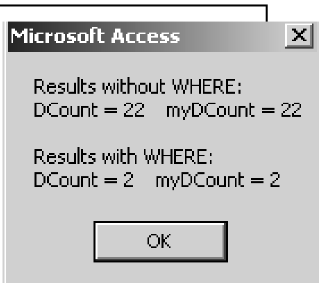
A call of: myDCount(“[Membership No]”, “[Membership]”)
has run-time strSQL of: "SELECT [Membership No] FROM Membership"

And a call of: myDCount(“[Membership No]”, “[Membership]”, “Lastname = ‘Locker’”)
has run-time strSQL of: "SELECT [Membership No] FROM Membership WHERE Lastname = 'Locker'"

Create a Click event for the cmdDCount button on your new form, with the code shown in Fig 3.4.3. This code calls the myDCount function twice, once without a criterion parameter and once with. It also makes identical calls to the Access DCount function. A message box then displays the two sets of results, to compare the results (each pair should have the same values!)

If you click on the cmdDCount button you should see the message box shown in Fig 3.4.3. Your totals may be different as your Membership table may have different numbers of records.

```
Private Sub cmdDCount_Click()

Dim VBACount As Integer          'results from Access functions
Dim VBACountFull As Integer
Dim intCount As Integer          'results from own functions
Dim intCountFull As Integer

    intCount = myDCount("[Membership No]", "[Membership]")
    VBACount = DCount("[Membership No]", "[Membership]")

   intCountFull = myDCount("[Membership No]", "[Membership]", "Lastname = 'Locker'")
   VBACountFull = DCount("[Membership No]", "[Membership]", "Lastname = 'Locker'")

    'compare VBA function with this one - should get the same result
    MsgBox "Results without WHERE:" & vbCrLf _
                   & "DCount = " & VBACount & "    " & "myDCount = " & intCount _
                   & vbCrLf & vbCrLf & "Results with WHERE:" & vbCrLf _
                   & "DCount = " & VBACountFull & "    " & "myDCount = " & intCountFull

End Sub
```



Microsoft Access

Results without WHERE:
DCount = 22   myDCount = 22

Results with WHERE:
DCount = 2   myDCount = 2

Fig 3.4.3 command button code to test the myDCount function with the message box showing the result

3.4.2 myDCount function – version 2

An alternative version of the code would be to use the aggregate function COUNT in the SQL, so that just one row is read into the Recordset, having a single field with the required count. Change your myDCount using the code shown in Fig 3.4.4.

```
Public Function myDCount(prmExpr As String, prmDomain As String, Optional prmCriteria As String) As Integer

Dim myDB As Database                       'declare Database variable
Dim rsData As DAO.Recordset                'declare Recordset variable
Dim strSQL As String                       'used for the SQL for the data
Const myTotalName = "TheTotal"     'used for the fieldname in the SQL

    Set myDB = CurrentDb        'specify database = assume the current database

    'create SQL to select count of all records, with default name using the Const
    strSQL = "SELECT COUNT(" & prmExpr & ") AS " & myTotalName & " FROM " & prmDomain

    'if a criterion is present, then add this to the SQL
    If IsMissing(prmCriteria) Or prmCriteria = "" Then   'will also be True if prmCriteria empty
        'do nothing
    Else
        strSQL = strSQL & " WHERE " & prmCriteria      ‘add the WHERE clause
    End If

    Set rsData = myDB.OpenRecordset(strSQL)   'get records using given SQL
    ‘code to check RecordCount has been deleted

    myDCountFull = rsData.Fields(myTotalName)       'return the required field

    rsData.Close       'close the Database...
    myDB.Close        '... and the Recordset

End Function
```

Fig 3.4.4 myDCount function version 2 with changes shown in bold font

This code is much simpler as there is no need to check the RecordCount. It is probably also more efficient as it reads just one record (which itself has just one field) into the Recordset. Version 1 required the SQL to read all the Membership records into the Recordset then the code itself read through all the records for the RecordCount to count them up.

At runtime, the SQL code in strSQL would look like (check this in the Debugger):

```
"SELECT COUNT([Membership No]) AS TheTotal FROM Membership WHERE Lastname = 'Locker'"
```

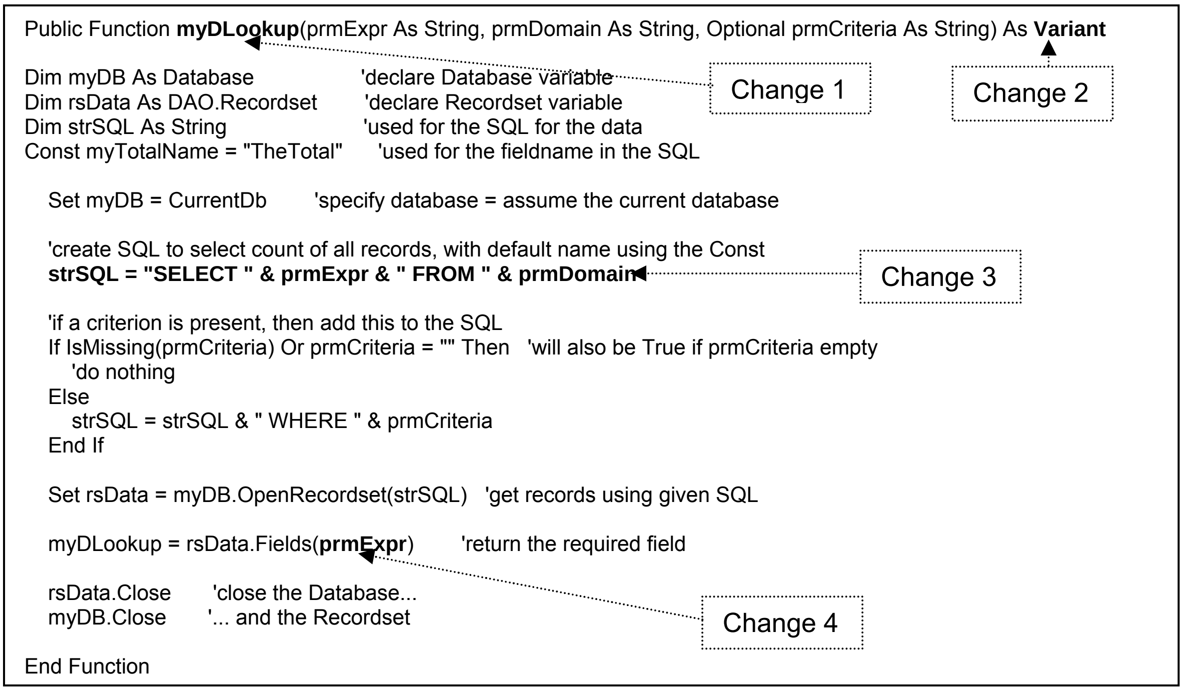
The result of this SQL statement will be a dataset with just one row, with just one field called TheTotal. This field will contain the count of the selected records. The field has been given a specific name so that it can then be referred-to by code in the statement:

```
myDCountFull = rsData.Fields(myTotalName)       'return the required field
```

If you click on your cmdDCount button on the testing form, you will now run the new version of the function, and should get exactly the same result as before.

3.4.3 myDLookup function – version 1

The Access built-in function DLookup which returns the contents of a named field in a table/query is called in an identical manner to DCount. A possible myDLookup function is shown in Fig 3.4.5. This was created very simply by copying-&-pasting version 2 of the myDCOunt function from Fig 3.4.5 and making the changes shown in bold font.



```
Public Function myDLookup(prmExpr As String, prmDomain As String, Optional prmCriteria As String) As Variant

Dim myDB As Database                'declare Database variable
Dim rsData As DAO.Recordset         'declare Recordset variable
Dim strSQL As String                'used for the SQL for the data
Const myTotalName = "TheTotal"      'used for the fieldname in the SQL

  Set myDB = CurrentDb      'specify database = assume the current database

  'create SQL to select count of all records, with default name using the Const
  strSQL = "SELECT " & prmExpr & " FROM " & prmDomain

  'if a criterion is present, then add this to the SQL
  If IsMissing(prmCriteria) Or prmCriteria = "" Then   'will also be True if prmCriteria empty
    'do nothing
  Else
    strSQL = strSQL & " WHERE " & prmCriteria
  End If

  Set rsData = myDB.OpenRecordset(strSQL)   'get records using given SQL

  myDLookup = rsData.Fields(prmExpr)       'return the required field

  rsData.Close      'close the Database...
  myDB.Close       '... and the Recordset

End Function
```

Fig 3.4.5 myDLookup function version 1, with differences from myDCount shown in bold

The changes that have been made here are:

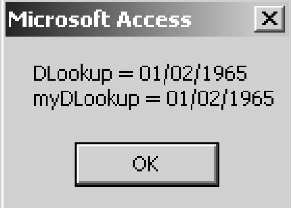
1. The function name is changed from myDCount to myDLookup.
2. The datatype of the return value is changed from Integer to Variant. The standard aggregate functions of COUNT, MAX, SUM, AVG, etc return a numeric value, so an Integer datatype was appropriate for myDCount. The myDLookup function will return the contents of a named field in a table/query, which could be of any datatype, so a Variant datatype is appropriate here.
3. The SQL has been changed, so that it is here of the format:
   SELECT <field name> FROM <table/query name> [WHERE…]
4. The Recordset field name is now the name of the table/query field requested by the calling code.

Create a click event for your cmdDLookup button with the code shown in Fig 3.4.6. When you click on the cmdDLookup button you should get the message shown in Fig 3.4.6. Change your cmdLookup_Click code to use different fields and different (or no) criteria.

At runtime, the SQL code in strSQL would look like (check this in the Debugger):

```
"SELECT [Date of Birth] FROM Membership WHERE [Membership No] = 15"
```

```
Private Sub cmdDLookup_Click()

Dim VBAData As Variant   'result from Access function
Dim varData As Variant     'result from own function

    VBAData = DLookup("[Date of Birth]", "Membership", "[Membership No] = 15")

    varData = myDLookup("[Date of Birth]", "Membership", "[Membership No] = 15")

    'compare VBA function with own function
    MsgBox "DLookup = " & VBAData _
            & vbCrLf _
            & "myDLookup = " & varData

End Sub
```



```
DLookup = 01/02/1965
myDLookup = 01/02/1965
```

Fig 3.4.6 command button code to test the myDLookup function with the message box showing the result

3.4.4 A common procedure for the myD functions

In section 3.4.3 you were instructed to create the code for your myDLookup function by copying-&-pasting code from elsewhere. But if you ever find yourself doing something like this you need to stop and think about using a common procedure. The main differences between the myDCount and the myDLookup functions are:

1. One uses aggregate SQL and the other doesn't.
2. One returns the named field from the aggregate SQL and the other returns the field specified by the calling code.

So, if the code knew which type of function was required, it should be able to take the appropriate action.

Add the code shown in Fig 3.4.7 to your Access code module.

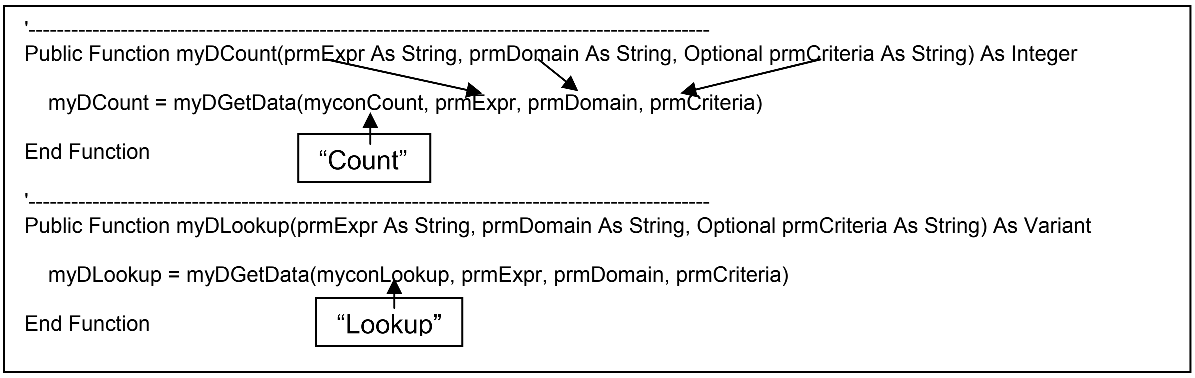
Points to note:

- All the parameters passed by the calling code are also required by the new procedure myDGetData.
- The procedure has an extra parameter as well, called prmType.
  - This parameter is to be one of the Private Const declarations at the start of Fig 3.4.7.
  - It cannot go at the end of the list of other parameters, as Optional parameters must be at the end, so I have put it at the beginning.
- The datatype of the return value is Variant, as this will cope with the Integer value required by aggregate functions such as myDCount and with the varying datatypes of the values to be returned by myDLookup.
- The Const and Dim statements are the same as before, with the exception of the new one for strFieldName. This is used to store either the name of the field requested by the calling code for myDLookup or the name of the total field in the aggregate SQL, and is used later when referencing the field in the Recordset.
- In order to generate the SQL, the code checks the value in prmType, and sets up either a simple SELECT SQL statement or an aggregate function SQL. The latter uses the value in prmType as the aggregate function in the SQL.
- The statement to return the required value now references strFieldName which earlier code has set to the required value.
- The rest of the code is the same as before.

```
Private Const myconLookup = "Lookup"
Private Const myconCount = "COUNT"

'-------------------------------------------------------------------------------------------------
Private Function myDGetData(prmtype As String, prmExpr As String, prmDomain As String, _
                        Optional prmCriteria As Variant) As Variant
'common module for use by all the 'myD' Domain Aggregate functions
'opens & closes the Database and Recordset; creates & uses SQL; returns the given field from the Recordset

Const myTotalName = "TheTotal"      'used for the fieldname for aggregate SQL

Dim myDB As Database                'declare Database variable
Dim rsData As DAO.Recordset         'declare Recordset variable
Dim strSQL As String                'the SQL for the Recordset
Dim strFieldName As String      'name of field in SQL

   Set myDB = CurrentDb        'specify database = assume the current database

   'start the SQL - two types - aggregate or lookup
   If prmtype = myconLookup Then
         strSQL = "SELECT " & prmExpr & " FROM " & prmDomain   'select fieldname from table/query
         strFieldName = prmExpr                                'fieldname for required contents
   Else   'assume this is an aggregate function
          'create SQL to select total of all records, with default name using the Const
         strSQL = "SELECT " & prmtype & "(" & prmExpr & ") AS " & myTotalName _
                       & " FROM " & prmDomain
         strFieldName = myTotalName      'default name in SQL for aggregate query
   End If

   'if a criterion is present, then add this to the SQL
   If IsMissing(prmCriteria) Or prmCriteria = "" Then   'will also be True if prmCriteria empty
      'do nothing
   Else
      strSQL = strSQL & " WHERE " & prmCriteria
   End If

   Set rsData = myDB.OpenRecordset(strSQL)   'get records using given SQL

   myDGetData = rsData.Fields(strFieldName)     'return the required field

   rsData.Close      'close the Database...
   myDB.Close       '... and the Recordset

End Function
```

Fig 3.4.7 Common myDGetData function to handle the DAO code for all the myD functions with code additional to previous code shown in bold

You can now simplify the code for myDCount and myDLookup as shown in Fig 3.4.8. Note that the parameters passed to the myD function by the calling code are passed through to myDGetData along with a further parameter (using the Consts) indicating the type of function required.



```
'-------------------------------------------------------------------------------------------------
Public Function myDCount(prmExpr As String, prmDomain As String, Optional prmCriteria As String) As Integer

   myDCount = myDGetData(myconCount, prmExpr, prmDomain, prmCriteria)

End Function
'-------------------------------------------------------------------------------------------------
Public Function myDLookup(prmExpr As String, prmDomain As String, Optional prmCriteria As String) As Variant

   myDLookup = myDGetData(myconLookup, prmExpr, prmDomain, prmCriteria)

End Function
```

Fig 3.4.8 Final versions of myDCount and myDLookup

The examples here are just for two of the functions. See Exercise 3.7.2.

3.5 The Object Browser

Access 2002 VBA Help provides the following information (enter *Object Browser* as VBA *Help* keywords):

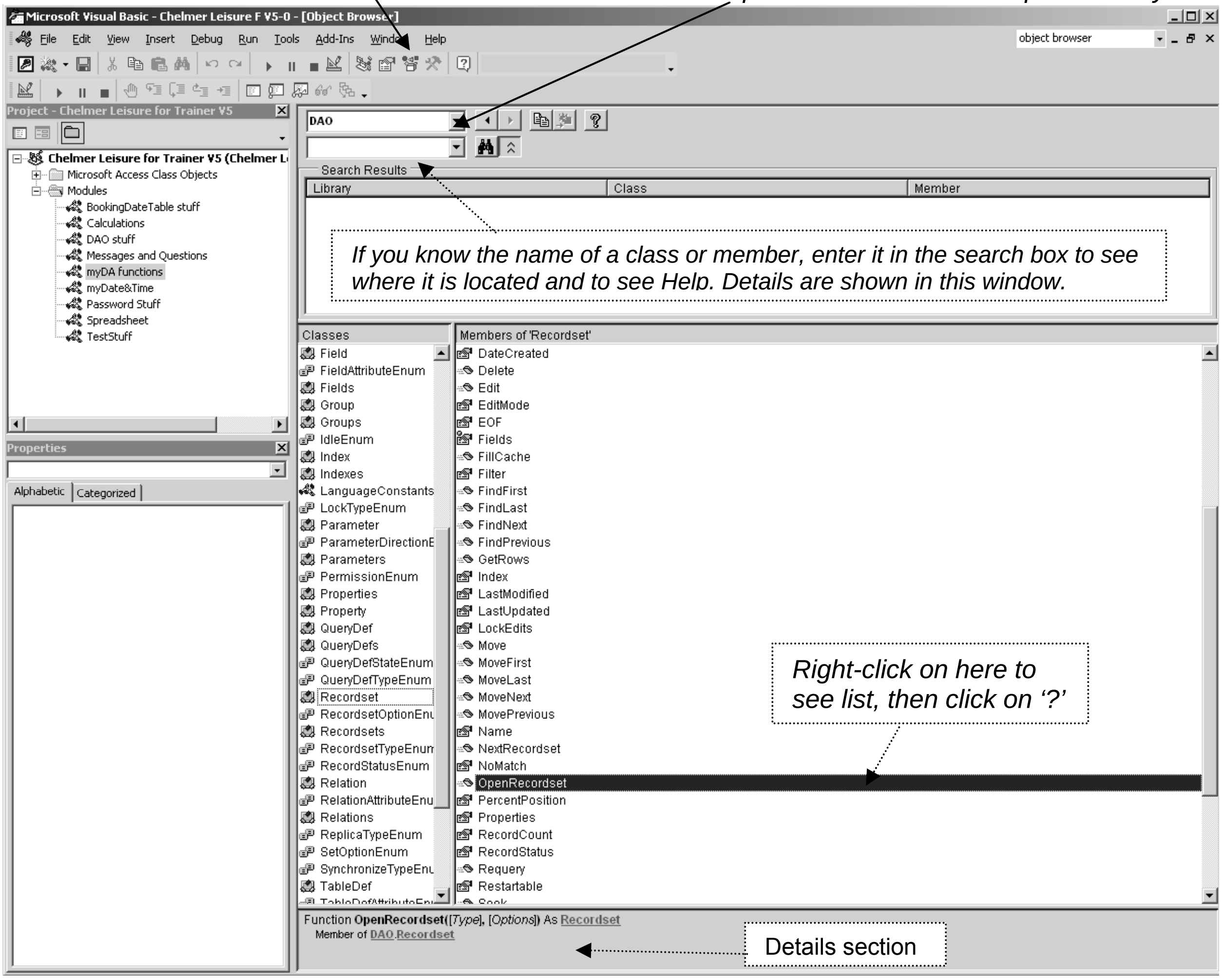
“The **Object Browser** allows you to browse through all available objects in your project and see their properties, methods and events. In addition, you can see the procedures and constants that are available from object libraries in your project. You can easily display online Help as you browse. You can use the **Object Browser** to find and use objects you create, as well as objects from other applications.
You can get help for the **Object Browser** by searching for **Object Browser** in Help.
To navigate the Object Browser

1. Activate a module.
2. From the **View** menu, choose **Object Browser** (F2), or use the toolbar shortcut.
3. Select the name of the project or library you want to view in the **Project/Library** list.
4. Use the **Class** list to select the class; use the **Member** list to select specific members of your class or project.
5. View information about the class or member you selected in the **Details** section at the bottom of the window.
6. Use the **Help** button to display the Help topic for the class or member you selected. “

So – if you open a code module you can bring up, and use, the screen shown in Fig 3.5.1. Click on F2 or the Object Browser icon.

Object Browser icon.

Drop down list to choose required library.



If you know the name of a class or member, enter it in the search box to see where it is located and to see Help. Details are shown in this window.

Right-click on here to see list, then click on ‘?’

Function **OpenRecordset**([*Type*], [*Options*]) As Recordset
Member of DAO.Recordset

Details section

Fig 3.5.1 The Object Browser

3.6 Summary

In order to use DAO code, remember the following:

- You will need to install the DAO library. See section 3.1.4.
- In order to specify a Database you must…
  - …declare a Database variable (a Dim statement) for each database that you are using.
  - …assign a value to the variable (associate a database with the variable) by using a Set statement.

  See section 3.2.1.
- You can use SQL statements to perform action queries on tables by using the Execute method. See section 3.2.2.
- In order to specify a Recordset you must…
  - …declare a Recordset variable (a Dim statement) for each set of table records that you want to read and/or write.
  - …associate the database and the table records with the Recordset via a Set statement for the OpenRecordset method with a SELECT SQL statement.

  See section 3.2.4.
- When reading through a Recordset, it is useful to use a loop until the EOF property is True.
  - The Recordset is positioned at the first record (if any) when it is first opened. If the Recordset is empty then the RecordCount property = 0, otherwise the RecordCount property = 1.
  - The methods MoveFirst, MoveLast, MoveNext and MovePrevious are used to navigate through the Recordset (and will fail if there is no current record; see Fig 3.2.9)
  - The RecordCount property contains the count of records read so far.
  - The Requery method will repopulate the Recordset and reset the RecordCount.
- When writing to a Recordset you need to code statements in the following order:
  - First specify the action required (AddNew, Edit or Delete).
  - Then make the amendments required. This may take several lines if you are putting values in several fields.
    - You can use the Fields property for this, or use a reference using the bang (!) operator.
  - Finally code the Update method. No changes are made to the table until this is executed.
- When the code has finished, remember to Close each Recordset and its associated Database.
  - The code doesn't always fail if you omit to do this, but sometimes it does (usually error 3211: "The database engine couldn't lock table <name> because it's already in use by another person or process. ").

3.7 Exercises

3.7.1 Create tables for bookings this year, split by type

Using DAO code read the Bookings table and write the records for all bookings made this year to two separate tables, one for Class bookings and one for Member bookings.

3.7.2 More MyD functions

Code the remaining myD functions such as myDMax, myDMin, myDAvg and myDSum.

See Appendix H.6 in the 'Getting Started' VBA Trainer for the full list of Access built-in Domain Aggregate functions.

You probably won't need DAO code, but may need to think carefully about how the other functions work.

PART 4 – WORKED EXAMPLE OF BOOKING PROCEDURE

REVIEW OF PART 4

In this part of the document you will see…

- …how to use DAO code to create and use a Booking diary page grid form.
  - Bookings are made by clicking on empty slots
  - Bookings are checked and deleted by clicking on booked slots
- …how to use Conditional Formatting to distinguish between booked and free slots.
- …how to use Access arrays:
  - All elements in an Access array must be of the same datatype.
  - You can specify the range to reference data elements, so that this makes sense in the context. Use an Integer datatype for the subscript.
  - Examples are shown of For…Next and Do Until…Loop for processing an array.
- …how to use DAO code:
  - To create a temporary table to store data then delete it when it is no longer needed.
    - Error-handling code is used to check if the temporary table already exists and take appropriate action if it does.
    - The DeleteObject method of DoCmd is used to delete the table.
  - Declare a variable of type Database to specify the database to be used. Set this to CurrentDb() for the Chelmer Leisure database.
  - Declare a variable of type Recordset to store data read from (and written to) a table.
  - Use the Database object OpenRecordset method to open a Recordset ready for use.
    - It is wise to use the Close method to close the Recordset and the Database when you have finished with it as this will also release locks on it.
  - Various Recordset methods have been used: RecordCount, AddNew, Fields, Update, MoveFirst, MoveNext, and EOF.
- …examples of using date and numeric fields as SQL WHERE criteria.
- …that the code shown can be adapted very simply to allow for bookings for fractions of an hour.

4.1 Introduction

In section 8.4 of the 'Getting Started' VBA Trainer you saw how to create a 'diary page grid' form to select a slot for a room to book, basing the form on a Crosstab query. Each cell of the form contained a '1' if the cell was booked and Null if it was free. By using embedded SQL, the Booking No and Membership or Class details for a booking could be extracted from the relevant table.

In this Part of this 'Further VBA' Trainer you will see an alternative method to create such a form, using DAO code. Here each cell will show the Booking No for booked slots.

This part of the document, showing some uses of DAOs and SQL, uses a fairly lengthy example that also illustrates some other useful things along the way. You will see how to create a BookingGrid form to make, check and delete bookings for the Chelmer Leisure Centre. See Fig 4.3.10 for the form; this is designed to simulate a page in a booking diary. This form is then used to make, check and delete bookings, with the user having to enter very little data via the keyboard. See section 4.7 for a diagram showing how all this fits together.

There will be 4 stages to the process of setting up the grid:

1. get the required booking table records into a Recordset
2. read the Recordset and store details into arrays
3. write the arrays out to a table
4. use this new table for the booking grid form.

These stages reorganise the Booking table data in a format that can then be used for a bound form.

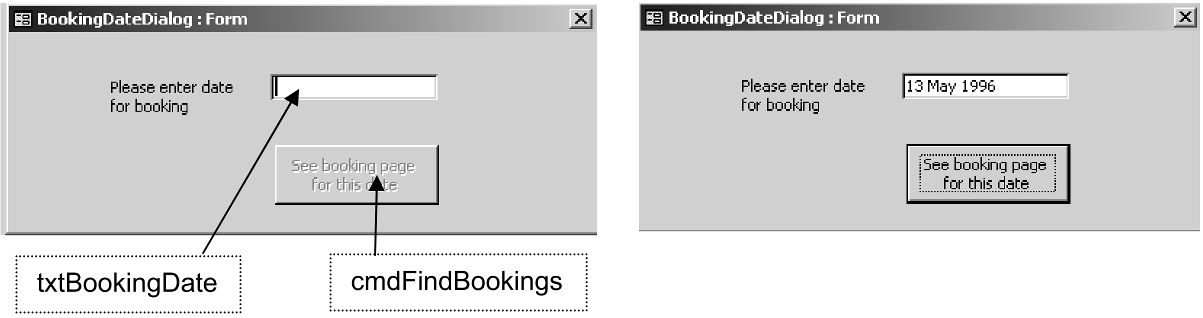
The table in Fig 4.1.1 summarises the remainder of this Part of the Trainer.

Sections	Purpose	Features demonstrated
4.3.1 – 4.3.2	Define the necessary variables	• Declarations for arrays, Databases, Recordsets, SQL strings. • Put the code to create the grid in a public procedure so that it can be used from more than one place. • Use of FOR loops to initialise arrays. • Look at array values in the Debugger.
4.3.3 – 4.3.5	Select records with just the required date and store in arrays	• Access a database table directly to read selected records into a Recordset. • Read the Recordset to put booking details into the appropriate array, using a DO loop. • Create a new table and write the array data out to this table using a FOR loop.
4.3.6	Create a form showing the bookings for the date as a grid (array).	• Create a form based on the new table. • Open the form once the table has been created. • Add the chosen date to the form. • Use conditional formatting to highlight booked slots.
4.4 - 4.5	Make, check and delete bookings directly via the grid, refreshing the grid to show the new information.	• Click on empty slots to create bookings . • Click on booked slots to check/delete bookings. • Call the booking grid module to redisplay the grid.
4.8	Show how to adapt the coding to booking for periods other than whole hours.	Very simple change to existing coding.

4.2 Create a dialog box to get the booking date

Create a simple form called BookingDateDialog as shown in Fig 4.2.1a with the code in Fig 4.2.1b. (You could replace the textbox by a calendar control if you wanted).

The code has a simple validation for the date and enables the command button only if the date value has been entered and is a valid date. It won't compile yet (you will get the error 'Sub or Function not defined') as the procedure myDisplayBookingGrid does not exist yet (but it will after the next section).


BookingDateDialog : Form
Please enter date for booking
See booking page for this date
txtBookingDate
cmdFindBookings
BookingDateDialog : Form
Please enter date for booking
See booking page for this date

13 May 1996

Fig 4.2.1a Simple BookingDateDialog form.

```
Option Compare Database
Option Explicit

'---------------------------------------------------------------------
Private Sub cmdFindBookings_Click()

   myDisplayBookingGrid txtBookingDate

End Sub

'--------------------------------------------------------------------------
Private Sub txtBookingDate_AfterUpdate()
'valid date entered, so enable the 'find' button

      cmdFindBookings.Enabled = True
      txtBookingDate = FormatDateTime(txtBookingDate, vbLongDate)

End Sub

'--------------------------------------------------------------------------
Private Sub txtBookingDate_BeforeUpdate(Cancel As Integer)
'check just in case user entered date then removed it
' if user does not enter date, the 'find' command button will not be enabled

   cmdFindBookings.Enabled = False    ‘assume invalid
   If IsNull(txtBookingDate) Then
      myDisplayWarningMessage "Please do not leave date blank"
      Cancel = True
   ElseIf Not IsDate(txtBookingDate) Then
      myDisplayWarningMessage "Please enter a valid date"
      Cancel = True
   Else
      Cancel = False     'valid date
   End If

End Sub
```

3). User clicks on command button to see booking grid form *(not created yet).*

2). if valid date entered, convert to Long Date format and enable command button.

1). Validate characters entered by user to ensure
(a) not left Null
(b) is valid date

Fig 4.2.1b Code for BookingDateDialog form.

4.3 Creating a BookingGrid form

Create a new Access module called BookingGridModule. All the code to create the BookingGrid form will be put into a Public procedure in this module, so that it can be used to create the grid initially and to refresh it after a change to the Booking table (see section 4.7).

Tip: use the Debugger as you test out the code, so that you can see what is going on and how the arrays are working.

4.3.1 Declaring variables

Open the new BookingGridModule code module. This should just contain the usual two default Option lines at the top. Add the declarations shown in Fig 4.3.1 after the Option Explicit line.

Points to note:

- The code makes the following assumptions:
  - Courts are booked by members only; Sports Halls and Fitness Suite are booked by Classes only (see page 240 of *McBride).*
  - All rooms can be booked between 09.00 and 20.00 (9 a.m. to 8 pm) inclusive, on all days of the week, for periods of one hour. The Centre closes at 21.00.
- All elements in a VBA array must be of the same datatype. Thus, rather than setting up a multi-dimensional array the code here uses lots of separate arrays. This structure also has the advantage that it should be easy to add/remove a room later.
  - The dtTimesArray will be used to display the booking time on the form. This will be initialised to the time; 09:00 to 20:00 (see next section).

  - The remaining arrays will be used to store the Booking No for the booking for that time on the required day. A numeric datatype could have been used (as these keys are AutoNumber, thus a numeric datatype), but this can only be initialised to zero. So a Variant datatype has been chosen instead so that this can be initialised to Null in order that empty entries can show as, and be checked for, blanks on the BookingGrid form. For more information on Variant datatypes see Access *Help*; note that it is more efficient to use a specific type where possible.
  - Each array corresponds to a column of the BookingGrid form shown in Figure 4.3.10.
  - The arrays could use '9 to 20' to reference the elements, as these references (also known as *subscripts* or *indexes*) correspond to the actual times of the day. However, the code here uses '0 to 11' as this allows the code to be adapted later for smaller intervals than one hour (see section 4.8). Rather than repeating this range many times, two constants myconStartTime and myconEndTime have been used; using such constants is good coding practice.
- The remaining fields are needed for various purposes which you will see in the following sections, as they are used.
- Your code for the BookingDateDialog form should now compile, as myDisplayBookingGrid now exists.

```
Option Compare Database
Option Explicit

'-----------------------------------------------------------------------------
'various public constants for general use
Public Const myconCourt1 = "Court 1"
Public Const myconCourt2 = "Court 2"
Public Const myconCourt3 = "Court 3"
Public Const myconFSuite = "Fitness Suite"
Public Const myconSHall1 = "Sports Hall 1"
Public Const myconSHall2 = "Sports Hall 2"

'various time constants for the booking intervals
Const myconActualStartTime = #9:00:00 AM#         'first booking time
Const myconActualEndTime = #8:00:00 PM#           'last booking time
Const myconTimeInterval = 1    'hour

'used for array range - saves repetition of numbers
Const myconStartTime = 0
Const myconEndTime = 11                           '09:00 to 20:00 inclusive, at hourly intervals
'have to work this out manually…
'could also use 9 to 20, but code here can be adapted better for smaller intervals with a range starting from 0

'arrays for booking data
Dim dtTimesArray(myconStartTime To myconEndTime) As Date            'entry for each time of day
Dim vCourt1Array(myconStartTime To myconEndTime) As Variant         'booking no for each time for Court1
Dim vCourt2Array(myconStartTime To myconEndTime) As Variant         'booking no for each time for Court2
Dim vCourt3Array(myconStartTime To myconEndTime) As Variant         'booking no for each time for Court3
Dim vFSArray(myconStartTime To myconEndTime) As Variant             'booking no for each time for Fitness Suite
Dim vSH1Array(myconStartTime To myconEndTime) As Variant            'booking no for each time for Sports Hall 1
Dim vSH2Array(myconStartTime To myconEndTime) As Variant            'booking no for each time for Sports Hall 2
Dim intCounter As Integer                                           'to reference array elements

'details required for accessing database records
Dim dbsThisDatabase As Database               'tell Access which database is to be used
Dim rstBookingTable As DAO.Recordset                    'copy of details from bookings table
Dim strSQL As String                          'for SQL statements
Dim rstTempTable As DAO.Recordset                       'records for the new table

Dim lngBookingNo As Long              'Booking No for the current record

'constants for type of booking action
Public Const myconBook = "Book"
Public Const myconDelete = "Delete"

'-----------------------------------------------------------------------------
Public Sub myDisplayBookingGrid(prmRequiredDate As Date)
   'code from Fig 4.3.2 goes here
End Sub
```

Comment these out if you have created the code from the 'getting Started' VBA Trainer Fig 8.1.1.

See section 4.8

Procedure called from cmdFindBookings button on BookingDateDialog form.

Fig 4.3.1 Variable Declarations for BookingGridModule code, plus Public procedure (as yet empty)

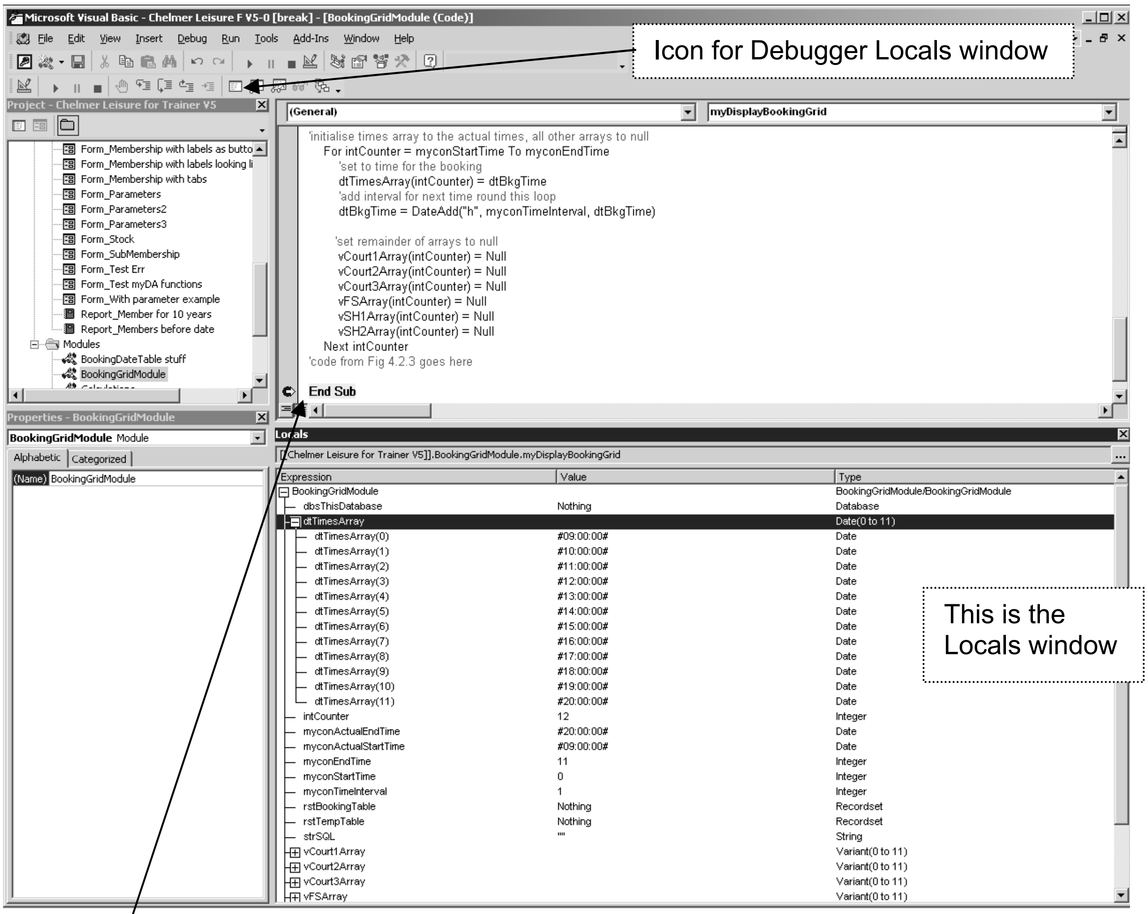
4.3.2 Initialising the arrays

Before the arrays can be used they need to be initialised to appropriate values. Add the code shown in Fig 4.3.2 to the new procedure myDisplayBookingGrid.

```
Dim dtBkgTime AS Date
   dtBkgTime = myconActualStartTime

'initialise times array to the actual times, all other arrays to null
   For intCounter = myconStartTime To myconEndTime        ‘for each element in the array
      'set to time for the booking
      dtTimesArray(intCounter) = dtBkgTime
      'add interval for next time round this loop
      dtBkgTime = DateAdd("h", myconTimeInterval, dtBkgTime)    ‘add one hour to get next slot

     'set remainder of arrays to null
      vCourt1Array(intCounter) = Null
      vCourt2Array(intCounter) = Null
      vCourt3Array(intCounter) = Null
      vFSArray(intCounter) = Null                                              see section 4.8
      vSH1Array(intCounter) = Null
      vSH2Array(intCounter) = Null
   Next intCounter
'code from Fig 4.3.4 goes here
```

Fig 4.3.2 Code to initialise arrays (goes in BookingGridModule)

By using the Debugger, you can check the values in the arrays; see Fig 4.3.3.


Microsoft Visual Basic - Chelmer Leisure F V5-0 [break] - [BookingGridModule (Code)]
File Edit View Insert Debug Run Tools Add-Ins Window Help
Project - Chelmer Leisure for Trainer V5
Form_Membership with labels as butto
Form_Membership with labels looking li
Form_Membership with tabs
Form_Parameters
Form_Parameters2
Form_Parameters3
Form_Stock
Form_SubMembership
Form_Test Err
Form_Test myDA functions
Form_With parameter example
Report_Member for 10 years
Report_Members before date
Modules
BookingDateTable stuff
BookingGridModule
Properties - BookingGridModule
BookingGridModule Module
Alphabetic Categorized
(Name) BookingGridModule
(General)
myDisplayBookingGrid
Locals

Icon for Debugger Locals window

```
'initialise times array to the actual times, all other arrays to null
   For intCounter = myconStartTime To myconEndTime
      'set to time for the booking
      dtTimesArray(intCounter) = dtBkgTime
      'add interval for next time round this loop
      dtBkgTime = DateAdd("h", myconTimeInterval, dtBkgTime)

     'set remainder of arrays to null
      vCourt1Array(intCounter) = Null
      vCourt2Array(intCounter) = Null
      vCourt3Array(intCounter) = Null
      vFSArray(intCounter) = Null
      vSH1Array(intCounter) = Null
      vSH2Array(intCounter) = Null
   Next intCounter
'code from Fig 4.2.3 goes here

End Sub
```

[Chelmer Leisure for Trainer V5]].BookingGridModule.myDisplayBookingGrid

Expression	Value	Type
BookingGridModule		BookingGridModule/BookingGridModule
dbsThisDatabase	Nothing	Database
dtTimesArray		Date(0 to 11)
dtTimesArray(0)	#09:00:00#	Date
dtTimesArray(1)	#10:00:00#	Date
dtTimesArray(2)	#11:00:00#	Date
dtTimesArray(3)	#12:00:00#	Date
dtTimesArray(4)	#13:00:00#	Date
dtTimesArray(5)	#14:00:00#	Date
dtTimesArray(6)	#15:00:00#	Date
dtTimesArray(7)	#16:00:00#	Date
dtTimesArray(8)	#17:00:00#	Date
dtTimesArray(9)	#18:00:00#	Date
dtTimesArray(10)	#19:00:00#	Date
dtTimesArray(11)	#20:00:00#	Date
intCounter	12	Integer
myconActualEndTime	#20:00:00#	Date
myconActualStartTime	#09:00:00#	Date
myconEndTime	11	Integer
myconStartTime	0	Integer
myconTimeInterval	1	Integer
rstBookingTable	Nothing	Recordset
rstTempTable	Nothing	Recordset
strSQL	""	String
vCourt1Array		Variant(0 to 11)
vCourt2Array		Variant(0 to 11)
vCourt3Array		Variant(0 to 11)
vFSArray		Variant(0 to 11)

This is the Locals window

Fig 4.3.3 Looking at array contents in the Debugger

- Set a breakpoint on the End Sub line in the procedure myDisplayBookingGrid.
- Open the BookingDateDialog form, choose any date and click on the cmdFindBookings button. The breakpoint will now cause the execution to stop on the breakpoint line and show the code window at this point.
- Click on the icon for the Debugger Locals window, then on the little + sign by *BookingGridModule* in the top part of the Locals window. This will show all the items that can be checked here. Click

on the + sign by dtTimesArray and see the values #09:00:00# to #20:00:00# in the array elements. Similarly, check the other arrays to see that they have Null in each element.

- If you wanted you could set a breakpoint earlier in the code and watch as the dtTimesArray values are entered in.
- Remove the breakpoint, close the code and form.

4.3.3 Stage 1 - get the booking table records

Add the code shown in Fig 4.3.4 to the myDisplayBookingGrid procedure.

```
'STAGE 1 - GET THE BOOKING TABLE RECORDS

'tell Access that we are using this database
   Set dbsThisDatabase = CurrentDb()

'select just those records that match the required date
   strSQL = "SELECT * FROM Bookings WHERE Date = DateValue(' " & prmRequiredDate & " ') "
   Set rstBookingTable = dbsThisDatabase.OpenRecordset(strSQL)
   '****testing only - to check code works so far
   If rstBookingTable.RecordCount <> 0 Then
      rstBookingTable.MoveLast      'read to end
   End If
   MsgBox "Bookings = " & rstBookingTable.RecordCount   'show count of records
   '****end of testing code
'code from Fig 4.3.6 goes here
```

Code for testing this stage.
To be removed after testing.

Fig 4.3.4 Code for stage 1, to read booking data into the Recordset

Brief explanation of code:

- Set dbsThisDatabase = CurrentDb() is the first use of DAO code (apart from variable declarations) in this Part. This line merely tells Access that the following DAO code refers to the current (i.e. your Chelmer Leisure) database. See section 3.2.1.
- The SQL statement assigned to strSQL consists of 3 parts:
  (i) "SELECT...FROM...WHERE...= DateValue(' "
  (ii) prmRequiredDate (the date parameter passed by the BookingDateDialog form)
  (iii) " ') "
  (i) & (iii) are partial SQL strings, (ii) puts the prmRequiredDate value in the string. The elements are joined together (concatenated) by &. The SQL thus selects all fields (*) from rows in the Bookings table in the Chelmer Leisure database, which have the booking date equal to the date the user has chosen on the BookingDateDialog form. If the date chosen was 13th May 1996, then the SQL will read

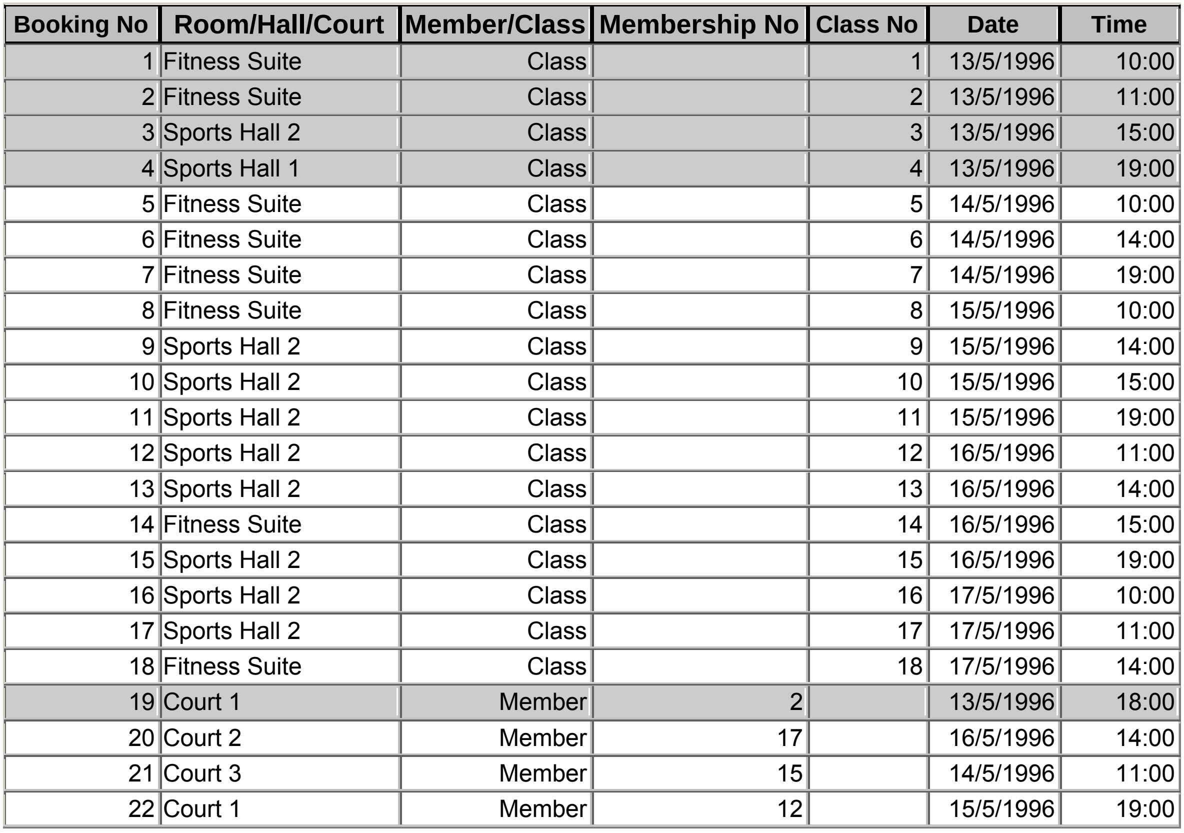
  "SELECT * FROM Bookings WHERE Date = DateValue(' 13/05/1996 ') "

  Use the Debugger to check the contents in prmRequiredDate and strSQL
  - It might seem logical to code

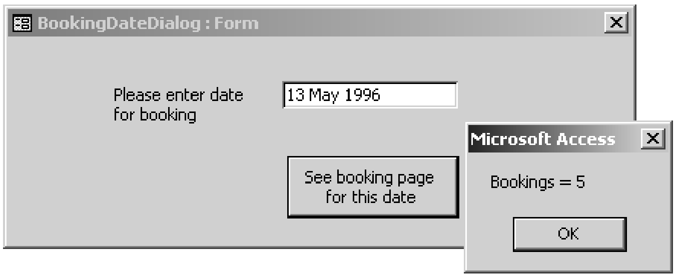
    strSQL = "SELECT * FROM Bookings WHERE Date = #" & prmRequiredDate & "#"

    for the SQL to select the bookings for the given date. But this code only works for certain dates. It will pick up records for dates with a day of 13 or above, but does not select those in the range 1-12; I have no idea why this is. Using the DateValue function seems to select all the dates.
- Set rstBookingTable = dbsThisDatabase.OpenRecordset(strSQL) opens the Recordset (see the variable definition in Fig 4.3.1). This statement uses the OpenRecordset method of the Database object and is where the SQL is executed. The records in this Recordset can now be investigated; this will be done in the next section.
- The following lines are merely there to test the code so far. When writing code for a complex process, it is useful to compile and test it in stages, rather than writing it all at once, attempting then to clear several compilation errors and then testing it all at once and trying to debug it.
- Fig 4.3.5 shows a suggested test plan for this stage.
- Finally, remember that you can position the cursor on any word in a line of code, press F1 and the *Help* system will then show you the relevant details.

Test No	Booking Date	Reason for test	Expected result
1	13/5/1996	Selection of bookings records, shown highlighted in table below.	5 records found
2	20/5/1996	No matching bookings records	No records found



Booking No	Room/Hall/Court	Member/Class	Membership No	Class No	Date	Time
1	Fitness Suite	Class		1	13/5/1996	10:00
2	Fitness Suite	Class		2	13/5/1996	11:00
3	Sports Hall 2	Class		3	13/5/1996	15:00
4	Sports Hall 1	Class		4	13/5/1996	19:00
5	Fitness Suite	Class		5	14/5/1996	10:00
6	Fitness Suite	Class		6	14/5/1996	14:00
7	Fitness Suite	Class		7	14/5/1996	19:00
8	Fitness Suite	Class		8	15/5/1996	10:00
9	Sports Hall 2	Class		9	15/5/1996	14:00
10	Sports Hall 2	Class		10	15/5/1996	15:00
11	Sports Hall 2	Class		11	15/5/1996	19:00
12	Sports Hall 2	Class		12	16/5/1996	11:00
13	Sports Hall 2	Class		13	16/5/1996	14:00
14	Fitness Suite	Class		14	16/5/1996	15:00
15	Sports Hall 2	Class		15	16/5/1996	19:00
16	Sports Hall 2	Class		16	17/5/1996	10:00
17	Sports Hall 2	Class		17	17/5/1996	11:00
18	Fitness Suite	Class		18	17/5/1996	14:00
19	Court 1	Member	2		13/5/1996	18:00
20	Court 2	Member	17		16/5/1996	14:00
21	Court 3	Member	15		14/5/1996	11:00
22	Court 1	Member	12		15/5/1996	19:00



Please enter date for booking 13 May 1996

Bookings = 5

Fig 4.3.5 Test plan for this stage, with Bookings data from McBride.
Records expected to be selected for test 1 highlighted.
Screen print of form with message for test 1.

4.3.4 Stage 2 - read Recordset details and put into arrays

Extend the code in Fig 4.3.4 to include the code shown in Fig 4.3.6. You can leave the Fig 4.3.4 testing code in if you want, or comment-it out or delete it.

Put a breakpoint on the IF statement in this new code. Open the BookingDateDialog form and choose the date 13/5/1996; as seen in Fig 4.3.5 there are 5 entries for this date in *McBride*. Click the cmdFindBookings button. Now you can use the Debugger to step through the code line by line, check on the values in various fields and watch as the bookings are entered into the arrays. The bulk of the code is just a loop for each record selecting the appropriate array in which to store details, and should be easy to follow.

The logic here can be summarised as:

```
If no records have been read
    Display 'no bookings' message
Else (at least one record found)
    Read first record in Recordset
    Do until end of Recordset
        Store details in appropriate array
        Get next record
    End Do
End if
```

The code here uses the Booking No to identify the booking in the arrays. Other possibilities are to use the Membership/Class No or the Class/Member name. You have to balance what you want to do, what the user may wish to see, how much room you have on the form/screen, and how your choice will affect the coding.

```
'STAGE 2 - PUT RECORDSET DETAILS INTO ARRAYS
  'RecordCount = 0 if no records found, or 1 otherwise
  If rstBookingTable.RecordCount = 0 Then
    myDisplayInfoMessage ("No bookings on " & prmRequiredDate)
  Else                                                          see section 4.8
    'position to first record in Recordset
    rstBookingTable.MoveFirst
    'take each row in the Recordset and add details to the appropriate array
    Do Until rstBookingTable.EOF        'loop until end of file
      'use booking time as reference for array element
      intCounter = Hour(rstBookingTable.Fields("time")) - Hour(myconActualStartTime)
      lngBookingNo = rstBookingTable.Fields("Booking No")  'booking no for array entry
      Select Case rstBookingTable.Fields("Room/Hall/Court")
        Case myconFSuite
          vFSArray(intCounter) = lngBookingNo
        Case myconSHall1
          vSH1Array(intCounter) = lngBookingNo
        Case myconSHall2
          vSH2Array(intCounter) = lngBookingNo
        Case myconCourt1
          vCourt1Array(intCounter) = lngBookingNo
        Case myconCourt2
          vCourt2Array(intCounter) = lngBookingNo
        Case myconCourt3
          vCourt3Array(intCounter) = lngBookingNo
        Case Else
          myDisplayWarningMessage "Invalid Room: ' " _
                & rstBookingTable.Fields("Room/Hall/Court") & " ' " _
                & vbCrLf & "Booking No: " & lngBookingNo
      End Select
      'get the next record
      rstBookingTable.MoveNext
    Loop
  End If
  'code from Fig 4.3.8 goes here
```

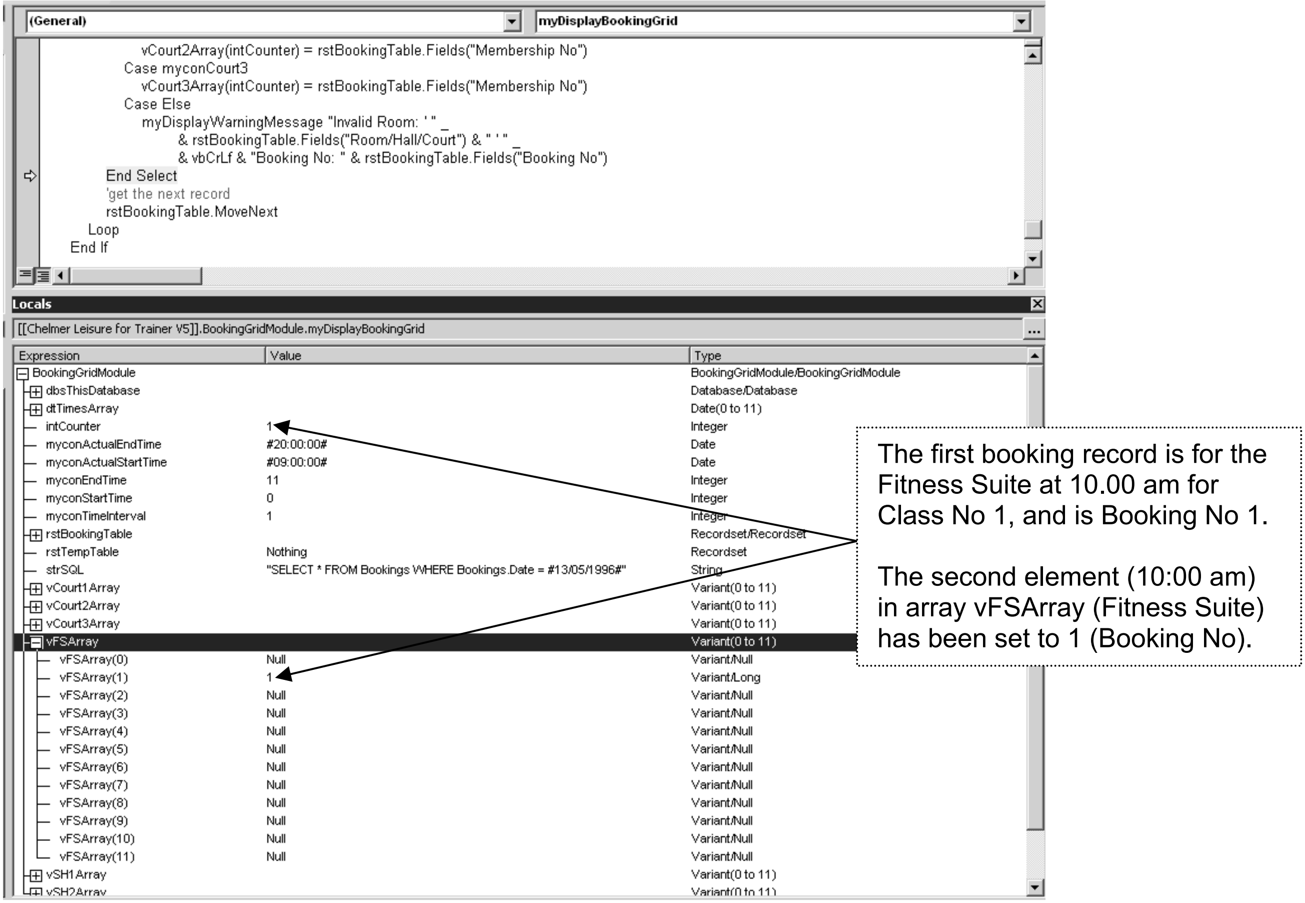
Fig 4.3.6 code to put booking details in the arrays.

Brief explanation of code (refer to section 3.2):

- The value in rstBookingTable.RecordCount is checked to see whether any records were selected.
- rstBookingTable.MoveFirst moves to the first record in the Recordset. This is not strictly necessary as the first record should already be the current record.
- rstBookingTable.EOF is initially False, then True when the end of the Recordset is reached or if no records were read, so is used to control the loop. (EOF = End Of File).
- Each record in the Recordset rstBookingTable is a copy of a row from the Bookings table. The fields in each record are referenced using the same names as used for each field (column) in the corresponding table. For example, rstBookingTable.Fields("Room/Hall/Court") refers to the value in the Room/Hall/Court field for the current record.
- intCounter is used to reference the array elements. It is calculated by subtracting the hour (9) of the start booking time (09:00) from the hour of the actual booking time. Thus a booking for 09:00 will give a value of 0, a booking for 10:00 will give a value of 1, and so on to a booking for 20:00 giving a value of 11.

- rstBookingTable.MoveNext moves to the next record in the Recordset. If there are no more records, EOF is set to True, and the loop will end.
- At the end of the loop, there should be an entry for each booking in the appropriate element of the array for the Room/Hall/Court.

Tip: whenever you are testing code that has a loop, test it in the Debugger. Otherwise you may get stuck in an infinite loop if your code is incorrect, and the only way of getting out of that is to end the task, and possibly lose any unsaved changes.



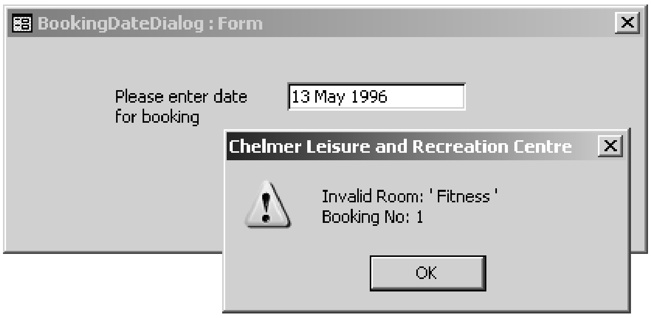
```
            vCourt2Array(intCounter) = rstBookingTable.Fields("Membership No")
          Case myconCourt3
            vCourt3Array(intCounter) = rstBookingTable.Fields("Membership No")
          Case Else
            myDisplayWarningMessage "Invalid Room: '" _
                & rstBookingTable.Fields("Room/Hall/Court") & "'" _
                & vbCrLf & "Booking No: " & rstBookingTable.Fields("Booking No")
        End Select
        'get the next record
        rstBookingTable.MoveNext
      Loop
    End If
```

```
[[Chelmer Leisure for Trainer V5]].BookingGridModule.myDisplayBookingGrid

Expression              Value                                                 Type
BookingGridModule                                                             BookingGridModule/BookingGridModule
  dbsThisDatabase                                                             Database/Database
  dtTimesArray                                                                Date(0 to 11)
  intCounter            1                                                     Integer
  myconActualEndTime    #20:00:00#                                            Date
  myconActualStartTime  #09:00:00#                                            Date
  myconEndTime          11                                                    Integer
  myconStartTime        0                                                     Integer
  myconTimeInterval     1                                                     Integer
  rstBookingTable                                                             Recordset/Recordset
  rstTempTable          Nothing                                               Recordset
  strSQL                "SELECT * FROM Bookings WHERE Bookings.Date = #13/05/1996#"  String
  vCourt1Array                                                                Variant(0 to 11)
  vCourt2Array                                                                Variant(0 to 11)
  vCourt3Array                                                                Variant(0 to 11)
  vFSArray                                                                    Variant(0 to 11)
    vFSArray(0)         Null                                                  Variant/Null
    vFSArray(1)         1                                                     Variant/Long
    vFSArray(2)         Null                                                  Variant/Null
    vFSArray(3)         Null                                                  Variant/Null
    vFSArray(4)         Null                                                  Variant/Null
    vFSArray(5)         Null                                                  Variant/Null
    vFSArray(6)         Null                                                  Variant/Null
    vFSArray(7)         Null                                                  Variant/Null
    vFSArray(8)         Null                                                  Variant/Null
    vFSArray(9)         Null                                                  Variant/Null
    vFSArray(10)        Null                                                  Variant/Null
    vFSArray(11)        Null                                                  Variant/Null
  vSH1Array                                                                   Variant(0 to 11)
  vSH2Array                                                                   Variant(0 to 11)
```

The first booking record is for the Fitness Suite at 10.00 am for Class No 1, and is Booking No 1.

The second element (10:00 am) in array vFSArray (Fitness Suite) has been set to 1 (Booking No).

Test No	Booking Date	Reason for test	Expected result
1	13/5/1996	Correct entries in arrays	5 correct Booking Nos in correct arrays, for correct time slots.
2	20/5/1996	No matching booking records	All room arrays have Null entries.
3	13/5/1996	Invalid Room/Hall/Court (record 1 for ‘Fitness Suite’ changed to read ‘Fitness’, then changed back after testing)	Error message: “Invalid Room: ‘Fitness’ Booking No 1”



Please enter date for booking: 13 May 1996

Invalid Room: ' Fitness '
Booking No: 1

Fig 4.3.7 Using the Debugger to see the execution of the code and a possible test plan
This Debugger screen print was taken after processing the first record in the Recordset for Test 1.
The BookingDateDialog form and error message are for test 3.

4.3.5 Stage 3 - Write arrays out to a new table

The code so far has recorded bookings details for the required date in the arrays. Now we need to write the details to a new (and temporary) table. Add the code shown in Fig 4.3.8 to the code so far.

```
Public Sub myDisplayBookingGrid(prmRequiredDate As Date)

On Error GoTo myDisplayBookingGrid_Err
:
:
:
'STAGE 3 - WRITE ARRAYS TO NEW (TEMPORARY) TABLE

    'create table, using SQL
    dbsThisDatabase.Execute "CREATE TABLE TempBooking (BookingTime DATE, " _
                            & "FSuite NUMBER, SHall1 NUMBER, SHall2 NUMBER, " _
                            & "Court1 NUMBER, Court2 NUMBER, Court3 NUMBER)"

    'open Recordset for new table
    Set rstTempTable = dbsThisDatabase.OpenRecordset("TempBooking")

    'for each element in arrays, write a record to the Recordset and then update the table
    For intCounter = myconStartTime To myconEndTime
        With rstTempTable
            .AddNew
            .Fields("BookingTime") = dtTimesArray(intCounter)
            .Fields("FSuite") = vFSArray(intCounter)
            .Fields("SHall1") = vSH1Array(intCounter)
            .Fields("SHall2") = vSH2Array(intCounter)
            .Fields("Court1") = vCourt1Array(intCounter)
            .Fields("Court2") = vCourt2Array(intCounter)
            .Fields("Court3") = vCourt3Array(intCounter)
            .Update
        End With
    Next intCounter
    rstTempTable.Close          'release locks or get run time error 3211
    dbsThisDatabase.Close
    'code to open booking grid form goes here – see end of section 4.3.6

Exit Sub

myDisplayBookingGrid_Err:
    Select Case Err
        Case 3010       'temp table already exists - can occur whilst testing
                        ' or if table not closed by code later
            DoCmd.DeleteObject acTable, "TempBooking"
            Resume   'carry on with statement where error occurred
        Case Else
            'carry on with Access error - display Err number plus message
            myDisplayWarningMessage (Err & "-" & Err.Description)
    End Select

End Sub
```

New line at top of procedure

Error-handling code to delete temporary table if it already exists, and carry on from point of failure.

Fig 4.3.8 Code to create, and add data to, new (temporary) table (goes in BookingGridModule)

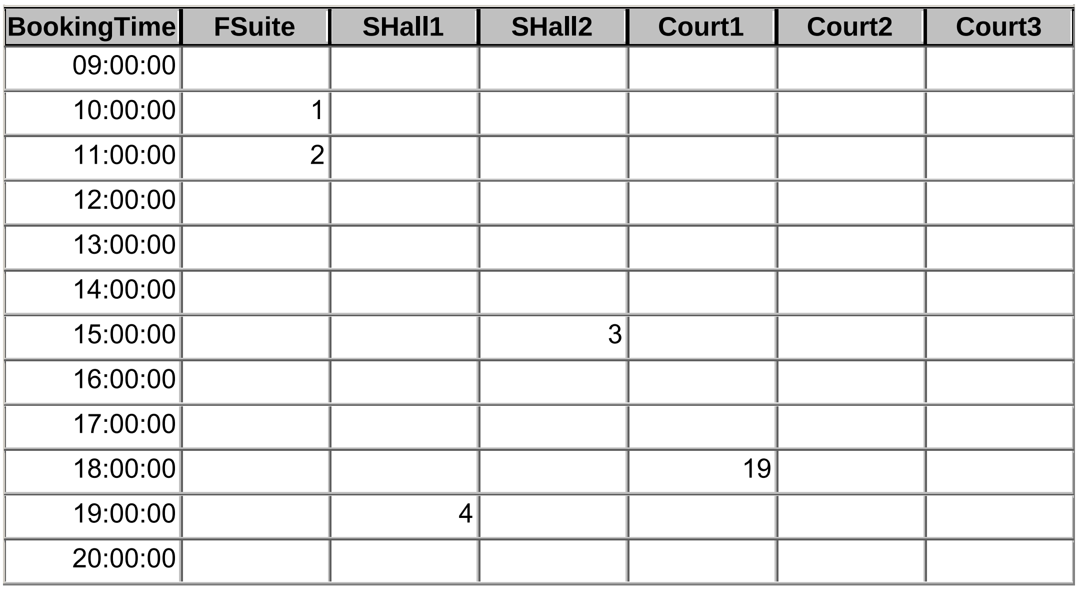
Brief explanation of code:

- This code creates a new (temporary) table in which to store the data that has been put into the arrays. A new line is added to the start of the procedure, to trap errors that may occur when handling this table. Errors are very likely to occur when testing new code.
- The main part of this new code (added after the existing code) starts by executing SQL to CREATE the new table. See the 'Getting Started' VBA Trainer Appendix G.1. The table is defined as having 7 columns, corresponding to those in Fig 4.3.10.
- The Recordset for the new table is opened and rows are written to it.
    - The With statement is used to simplify the coding.
    - The AddNew method is used to indicate that a new row is to be added to the new table (see section 3.3 for an earlier example).
    - Each field in the new record/row has the appropriate value entered into it, from the corresponding array entry, using the Fields method.
    - The Update method is then used to add the record/row to the new table.

  - The Recordset is then closed. This removes locks from the data so the table can be deleted to refresh the booking grid later. Failure to remove locks can cause run-time error 3211.
- There will be 12 rows in the table, corresponding to times 09:00 to 20:00 inclusive.
- The error-handling code looks for error 3010, as this error will occur if you run the code again at this stage when the temporary table already exists. The code simply deletes the table then carries on as though nothing untoward had happened. See Appendix J in the ‘Getting Started’ VBA Trainer for some useful DoCmd methods. Look at VBA help for the different forms of the Resume statement.

Now open the BookingDateDialog form again, enter the date 13/5/1996 and click the cmdFindBookings button. Nothing will appear to happen as everything is still going on behind the scenes. Go and look at the tables for the database, and there should be a new table called TempBooking, which should have the *McBride* data shown in Fig 4.3.9.

Test this stage with tests 1 and 2 from Stage 2.
Compare the format of the table with the CREATE TABLE SQL in Fig 4.3.8.



BookingTime	FSuite	SHall1	SHall2	Court1	Court2	Court3
09:00:00						
10:00:00	1					
11:00:00	2					
12:00:00						
13:00:00						
14:00:00						
15:00:00			3			
16:00:00						
17:00:00						
18:00:00				19		
19:00:00		4				
20:00:00						

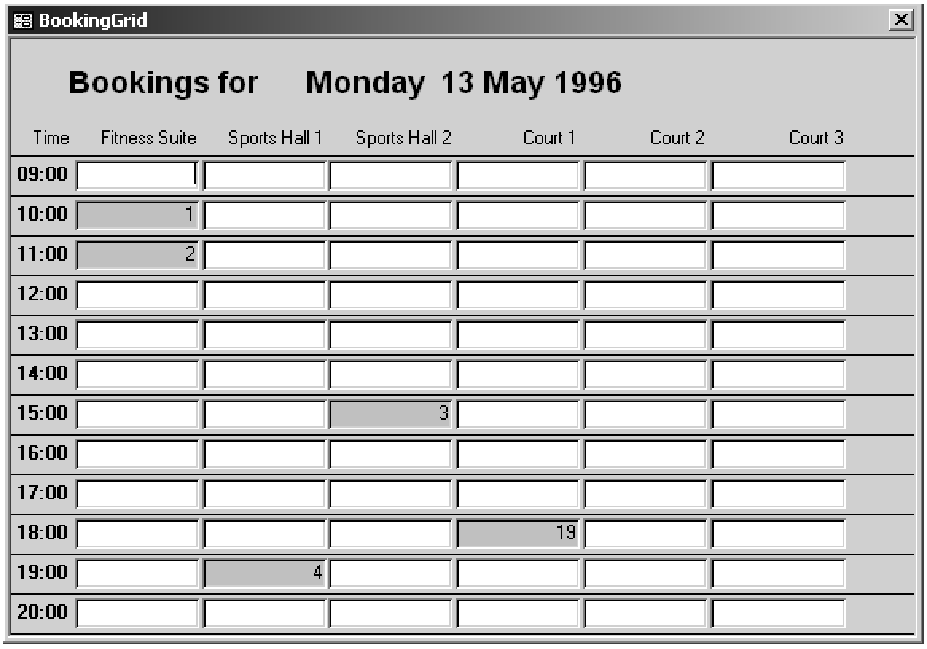
Fig 4.3.9 Data in TempBooking table for booking date of 13/5/1996
Check this with the Bookings data in Fig 4.3.5.

4.3.6 Stage 4 - Create the BookingGrid form

Now you can see that you have a table that looks like a diary page entry in a manual booking system. Using this new table, create a tabular form (use wizards) to display the table data. Call the form BookingGrid. Compare this to the form in Fig 8.4.10 of the ‘Getting Started’ VBA Trainer.

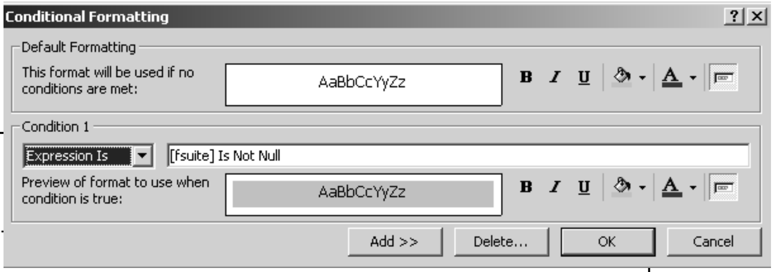
See Fig 4.3.10 for form improvements and code:

- Set the booking date in the header.
  - Create two fields called txtDate and txtDayName, and add the code shown.
- Format the Time field.
  - Set the Locked property to Yes, the Format to Short Time, plus other formats as appropriate.
- Improve the column headers (SHall1 = Sports Hall 1, etc).
- Remove navigation buttons, record selection indicators and scroll bars.
- Set the form AllowAdditions property to No (this removes the extra space for a new record at the end of the form).
- Use conditional formatting to set the booked slots to grey.
- Do any other tidying up that you wish to do to improve the general appearance of the form.



Bookings for Monday 13 May 1996

Time	Fitness Suite	Sports Hall 1	Sports Hall 2	Court 1	Court 2	Court 3
09:00						
10:00	1					
11:00	2					
12:00						
13:00						
14:00						
15:00			3			
16:00						
17:00						
18:00				19		
19:00		4				
20:00						



This format will be used if no conditions are met:

Condition 1: Expression Is [fsuite] Is Not Null

Preview of format to use when condition is true:

```
Option Compare Database
Option Explicit

'------------------------------------------------------
Private Sub Form_Load()
'put date for booking in the form header

Dim dtDate As Date

    dtDate = Forms!BookingDateDialog!txtBookingDate
    txtDate = FormatDateTime(dtDate, vbLongDate)
    txtDayName = (WeekdayName(Weekday(dtDate), , vbSunday))
    FSuite.SetFocus      'otherwise Focus may go to first time field

End Sub
```

Fig 4.3.10 BookingGrid form and initial code,
plus conditional formatting (do each one separately) for the booked slots.

Almost there - add the following code to the BookingGridModule to just before Exit Sub:

```
'STAGE 4 - OPEN BOOKINGGRID FORM TO DISPLAY BOOKINGS FOR DATE
    DoCmd.OpenForm "BookingGrid"
```

Now try opening the BookingDateDialog form and entering a date.
The BookingGrid form is displayed (at last!)
The user can close the BookingGrid form, the BookingDateDialog form will still be open, and the user can enter another date if wished, click on the cmdFindBookings button again and see the BookingGrid form again.

4.3.7 Delete the temporary table

The BookingGrid form is bound to the temporary table TempBooking, so the code to delete the table cannot be executed by that form module. So, add the code for a Close event as shown in Fig 4.3.11 to the module for the BookingDateDialog form. Now, when this form is closed the temporary table will be deleted.

If you split your database into front- and back-ends, the TempBooking table must be in the front-end, as each user will be checking bookings for different dates and using their own data in this table.

```
Private Sub Form_Close()
'delete the temporary table

    DoCmd.DeleteObject acTable, "TempBooking"

End Sub
```

Fig 4.3.11 Close event for BookingDateDialog form

4.4 Make a booking

Looking at the BookingGrid form in Fig 4.3.10, you can see which time-slots for a room are free; these are the slots that are empty and unshaded. It is a pretty simple process to create a Click event for each slot, check if the slot is empty then call up a Bookings form. This process works in the same way as shown in section 8.4.6 of the ‘Getting Started’ VBA Trainer.

4.4.1 Create and use text boxes for parameter values on the BookingGrid form

Create the three ‘hidden field’ text boxes called txtRoomHallCourt, txtMemberClass and txtDate on your BookingGrid form as shown in Fig 8.4.16 of the ‘Getting Started’ VBA Trainer.

Create two further ‘hidden field’ textboxes called txtType (to store whether the action is a new booking or a deletion of an existing booking) and txtNo (for the Booking No of an existing booking).

Add the code shown in Fig 4.4.1.

```
Private Sub Court1_Click()
'put values in hidden fields on form.
    myHiddenFields myconCourt1, myconMember
'check if slot free or booked and take appropriate action.
    myCheckSlot [Court1]
End Sub
'--------------------------------------------------------------------------
Private Sub FSuite_Click()
'put values in hidden fields on form.
    myHiddenFields myconFSuite, myconClass
'check if slot free or booked and take appropriate action.
    myCheckSlot [FSuite]
End Sub
'-----------------------------------------------------------------
Private Sub myCheckSlot(prmSlot As TextBox)
'see if slot is free or booked.
    If IsNull(prmSlot) Then      'is the slot Null?
        txtType = myconBook
        DoCmd.OpenForm "Bookings FVBA", , , , acFormAdd
    Else
        txtType = myconDelete
        txtNo = prmSlot
        DoCmd.OpenForm "Bookings FVBA", , , "[Booking No] = " & txtNo
    End If
End Sub
 '-----------------------------------------------------------
Private Sub myHiddenFields(prmRoomHallCourt As String, prmMemberClass As Boolean)
'called when user clicks on a slot
'copies the information to hidden fields on the form, to be used by booking and deletion processes.
    txtRoomHallCourt = prmRoomHallCourt
    txtMemberClass = prmMemberClass
    txtTime = BookingTime
End Sub
```

Member booking

Class booking

Common procedure to check the slot and put values in hidden fields.

Open the Bookings form in the required mode for the action.
Add: ready for new record
Delete: filter to see record with required Booking No.

Common procedure to put values in hidden fields

Fig 4.4.1 Code to click on free slots on BookingGrid form and make bookings

This code is basically the same as that from section 8.4.6.1 of the ‘Getting Started’ VBA Trainer. The field names and the name of the Bookings form (see next section) are different.

Bookings for the remaining slots can be coded quite easily, following the examples shown in Fig 4.4.1.

4.4.2 Create a Bookings form and code

Now create a simple Bookings Form (perhaps copy the one from section 8.4.6.2 of the ‘Getting Started’ VBA Trainer); see Fig 4.4.3. Mine is called ‘Bookings FVBA’ (see the highlighted name in Fig 4.4.1).

Set the code module for the Bookings form to have the code shown in Fig 4.4.2.

```
Option Compare Database
Option Explicit

'-----------------------------------------------------------------
Private Sub Form_Load()
'copy known information into Bookings form from hidden fields on BookingGrid form

'check for member or class booking
    If Forms![BookingGrid]!txtMemberClass = myconMember Then
        [Membership No].Visible = True
        [Class No].Visible = False
        [Membership No].SetFocus
    Else
         [Membership No].Visible = False
         [Class No].Visible = True
         [Class No].SetFocus
    End If

'set form up for add or delete
    If Forms!BookingGrid!txtType = myconBook Then
        'show appropriate buttons
        cmdDelete.Visible = False
        cmdConfirm.Visible = True
        cmdClose.Visible = False
        'put booking values in record - user only adds member/class no
        [Room/Hall/Court] = Forms![BookingGrid]!txtRoomHallCourt
        [Member/Class] = Forms![BookingGrid]!txtMemberClass
        [Date] = Forms![BookingGrid]!txtDate       'this is the form header date
        [Time] = Forms![BookingGrid]!txtTime
    Else    ‘is for a deletion
        Code from Fig 4.5.1 goes here
    End If

End Sub
```

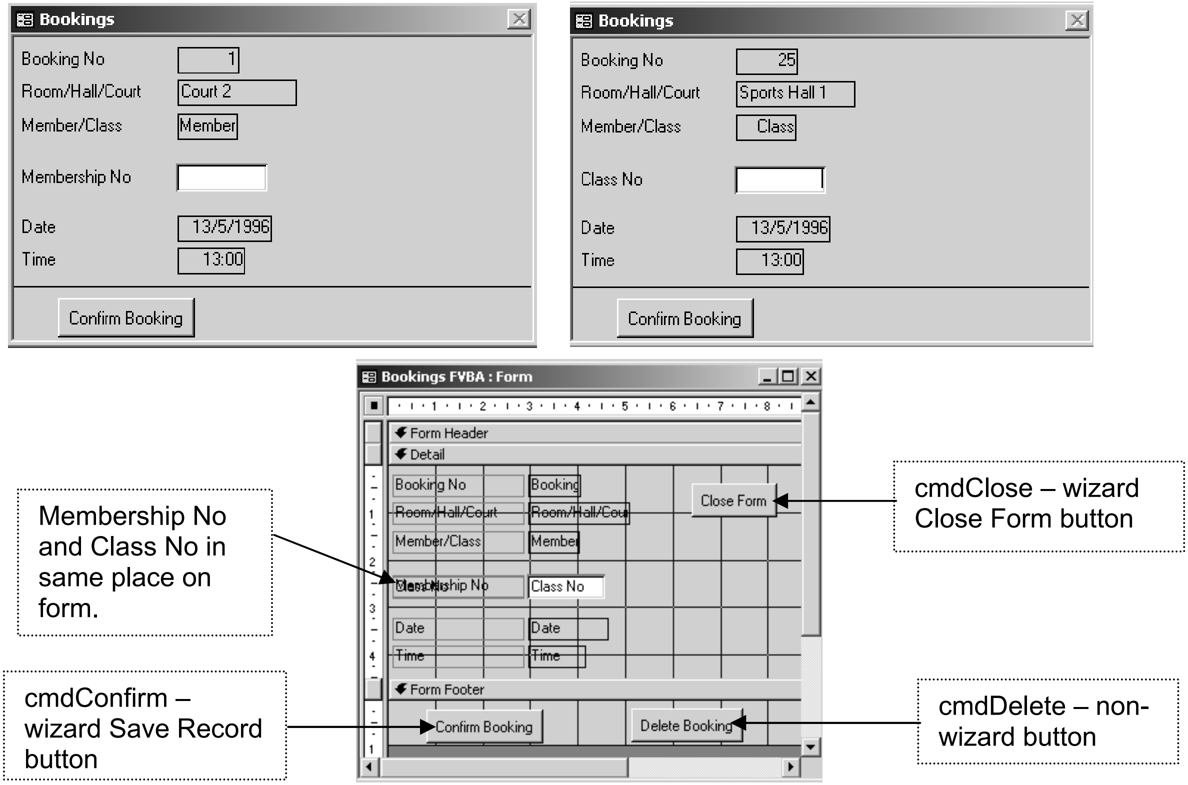
Show the appropriate field for the user to enter the Membership/Class No

Set rest of form up ready for a booking. The Booking No will be put there by Access. Only the cmdConfirm button will be visible.

Fig 4.4.2 Code for simple Bookings FVBA form

Explanation of code in Fig 4.4.2:

- The Form_Load event:
    - This checks information in the txtMemberClass field on the BookingGrid form to ensure that either the Membership No or the Class No field is showing on the Bookings form. The cursor is positioned in the field ready for the user to enter the required value.
    - If the value in txtType is for a booking (i.e. the user has clicked on a free slot) the values from the other hidden fields are put in the form.
    - The Confirm Booking button is made visible.
    - The Close and Delete buttons are made invisible.
    - This is much the same as in the ‘Getting Started’ VBA Trainer.



Booking No 1
Room/Hall/Court Court 2
Member/Class Member
Membership No
Date 13/5/1996
Time 13:00

Booking No 25
Room/Hall/Court Sports Hall 1
Member/Class Class
Class No
Date 13/5/1996
Time 13:00

Membership No and Class No in same place on form.

cmdConfirm – wizard Save Record button

cmdClose – wizard Close Form button

cmdDelete – non-wizard button

Fig 4.4.3 Simple Bookings form showing a Member booking (top left), a class booking (top right), and in design view showing command buttons.

```
Private Sub cmdClose_Click()
'wizard code to close form
On Error GoTo Err_cmdClose_Click

'close and reopen BookingGrid form to refresh it and show the new booking.
    DoCmd.Close acForm, "BookingGrid"
    myDisplayBookingGrid Forms![BookingDateDialog]!txtBookingDate
    DoCmd.Close acForm, "Bookings FVBA"    'must specify name if more than one Close statement

Exit_cmdClose_Click:
    Exit Sub

Err_cmdClose_Click:
    MsgBox Err.Description
    Resume Exit_cmdClose_Click

End Sub

'-----------------------------------------------------------------------
Private Sub cmdConfirm_Click()
On Error GoTo Err_cmdConfirm_Click

'wizard code to save record; can't use DAO code as form is bound to the table, so will then get two saves
    If myYesNoQuestion("Confirm this booking?") = vbYes Then
    'wizard code to save
        DoCmd.DoMenuItem acFormBar, acRecordsMenu, acSaveRecord, , acMenuVer70
    Else
        Undo     'form will close without saving
    End If
    cmdClose_Click     'automatically close the form

Exit_cmdConfirm_Click:
    Exit Sub

Err_cmdConfirm_Click:
    MsgBox Err.Description
    Resume Exit_cmdConfirm_Click

End Sub
```

Wizard code to close the form.
Additions shown in bold.
Button is not available for form opened in Add mode, but code is called by cmdConfirm button.

Wizard code to add the booking record.
Additions shown in bold.
Calls cmdClose code to close the form automatically

Fig 4.4.4 Code for Close and Confirm buttons on Bookings form

Explanation of code in Fig 4.4.4:

- cmdClose_Click is wizard code for a command button to close the form.
    - This button is not available to the user, so that the user can only exit by clicking the cmdConfirm button and confirming or cancelling the booking.
        - I cannot work out how to trap unsaved changes if the user clicks on a Close button; see the 'Getting Started' VBA Trainer sections 2.5.2 and 2.7.4.
    - The code in bold has been added to the wizard code. When the form is closed, the booking is added to the Bookings table and the BookingGrid form is refreshed (by closing it then recreating it) so that the new booking now shows on the grid. Try testing this out - see the new booking in the BookingGrid and the Bookings table after a booking is made.
    - Note that the wizard code to close this form now has the form name added to it as Access seems to need this if there is more than one Close statement.

- cmdConfirm_Click is wizard code for a command button to save the record.
    - The user is first asked whether or not he/she wishes to confirm the booking. If the reply is 'Yes' then the wizard line to save the code is executed, otherwise the changes are undone.
        - It might be useful to provide confirmation messages of the form "Booking *99* for *Room* on *Date* at *Time* saved OK" and "This booking cancelled" (where the items in italics are the relevant values for the booking).
        - On reflection, button text such as 'confirm or cancel booking' might be more informative.
    - The cmdClose_Click event is then called to close the form.
    - The form is bound to the Bookings table. If the addition of the new record was done by DAO code, the booking will be saved twice (each with a different Booking No), once by the DAO code and once automatically by Access. It is therefore best here not to use DAO code to save a record on a bound form. If you wanted to use DAO code (such as that shown in Fig 4.3.8) then use an unbound form so that Access will not do an automatic save and you are in control of the save event.
        - Using a bound form with an AutoNumber key has the advantage that the Booking No is shown on the form and the user can be informed of this for reference.

Test No	Action	Reason for test	Expected result
1	Click on free slot	• Check appearance of Booking form.	• Booking form shows correct booking details and only the Confirm button. • Cursor positioned in Membership No or Class No.
		• Click on Confirm button, say Yes to save.	• Booking form closes automatically. • BookingGrid form now shows booking. • Booking correctly shown in Bookings table.
2	As test 1 for all rooms and all times	• Repeat test 1 – code works correctly for all rooms and times.	• As test 1.
3	Click on free slot	• Click on Confirm button, say No to save.	• Booking form closes automatically. • BookingGrid form does not show booking. • No new booking shown in Bookings table.
4	Click on booked slot	• Check appearance of Booking form.	• Booking form shows correct booking details.
		• Click on Close button.	• Booking form is then closed (ignore buttons for now).

Fig 4.4.5 Possible test plan for BookingGrid and Bookings forms code so far.

4.5 Check/Delete a booking

In section 8.4.7 of the 'Getting Started' VBA Trainer deletion of bookings was all coded within the click event for a blank slot on the Bookings Crosstab form. That was fine there as the form was refreshed simply by using the Requery method to requery the data for the form. Here, however, we have to refresh the BookingGrid form by closing and recreating the table upon which the form is bound, so the delete process cannot be the same as before.

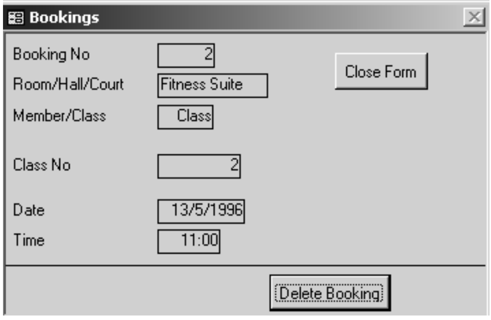
But we have the Bookings form from the previous section, 4.4, of this document, so this form can be used very simply to show the booking details and allow the user the option of simply closing the form or deleting the booking. This thus allows the user to check the booking details (to see which member or class has made the booking) and, optionally, to delete the booking. Look back at the code for Fig 4.4.1, in particular the line that opens the Bookings form if the user clicks on a booked slot:

```
DoCmd.OpenForm "Bookings FVBA", , , "[Booking No] = " & txtNo
```

This line opens the form with a WHERE condition for a filter to show only the booking with the Booking No shown in the booked slot. This will also show the Membership No or Class No as appropriate. You should have noticed this from test 4 of Fig 4.4.5.

You should also have noticed that the cursor is positioned in the Membership/Class No field and that this field looks as though it is ready for data entry, which is not appropriate here. The cmdDelete and cmdClose buttons may, or may not, be visible, and the cmdConfirm button probably is visible which is not appropriate here either. The code in Fig 4.5.1 will rectify these items.

```
'set form up for add or delete
  If Forms!BookingGrid!txtType = myconBook Then
    Code as in Fig 4.4.2
  Else    'is for a deletion
    'show appropriate buttons
    cmdDelete.Visible = True
    cmdConfirm.Visible = False
    cmdClose.Visible = True       'if user merely wants to view member/class details
    cmdDelete.SetFocus
    'set properties to show that fields are not for data entry
    With [Class No]
      .BackStyle = 0       'transparent
      .SpecialEffect = 0   'flat
      .Locked = True
    End With
    With [Membership No]
      .BackStyle = 0       'transparent
      .SpecialEffect = 0   'flat
      .Locked = True
    End With

  End If
```



Bookings

Booking No	2
Room/Hall/Court	Fitness Suite
Member/Class	Class
Class No	2
Date	13/5/1996
Time	11:00

Fig 4.5.1 Code for Bookings FVBA Form_Load event to set form up to check/delete a booking
Code in bold shows code to be added to that in Fig 4.4.2
Figure also shows screenprint of Bookings for in check/delete mode.

Explanation of code in Fig 4.5.1:

- The cmdConfirm button is hidden and the cmdDelete and cmdClose buttons now show. The cursor is positioned on the cmdDelete button.
- The properties for the Membership No and Class No fields are set to match the rest of the fields.
- The user now cannot change any of the data, and must click on either…
  - o …the cmdClose button. This will close the form without changing anything, although it will still refresh the BookingGrid form.
  - o …the cmdDelete button. See Fig 4.5.2 for the code for this.

Now add the code in Fig 4.5.2 to your Bookings FVBA form and link it to your cmdDelete button.

```
Private Sub cmdDelete_Click()
'delete the booking shown on the form
'could use wizard, but this shows how to do it with DAO code

Dim strSQL As String
Dim strWhere As String
Dim dbsThisDatabase As Database

  If myYesNoQuestion("Delete this booking?") = vbYes Then
    Set dbsThisDatabase = CurrentDb
    strSQL = "DELETE * FROM Bookings WHERE [Booking no] = " & [Booking No]
    dbsThisDatabase.Execute strSQL
    dbsThisDatabase.Close
    'close form
    cmdClose_Click
  End If

End Sub
```

Fig 4.5.2 Code in Bookings FVBA form to delete a booking

Explanation of code in Fig 4.5.2:

- The user is first asked if they really want to delete the booking. This just for safety in case they click on the button in error.
    - If the reply is 'Yes' then DAO code is used to delete the record.
        - The button could have been set up via the wizard to delete a record, but DAO code does not cause a conflict with Access's own checks here (as it does when adding a record on a bound form; see near end of section 4.4.2) so DAO code has been shown here to demonstrate how to do it.
    - If the reply is 'No' then nothing happens. The form is not closed, though it could be if that is what is wanted. This method puts the user back to the form and keeps them in control.
    - A message to inform the user that the booking has been deleted/retained might be useful..

Test No	Action	Reason for test	Expected result
1	Click on booked slot	• Check appearance of Booking form.	• Booking form shows correct booking details plus the Delete and Close buttons. • Member/Class No is flat etc. • Cursor positioned on Delete button.
		• Click on Delete button, say No to delete then click on Close.	• Booking form is not closed. • BookingGrid form still shows booking. • Booking still shown in Bookings table.
2	Click on booked slot	• Click on Delete button, say Yes to delete.	• Booking form closes automatically • BookingGrid form does not show booking. • Booking not shown in Bookings table.
3	As test 2 for all rooms and all times	• Repeat test 2 – code works correctly for all rooms and times.	• As test 2.
4	Click on booked slot	• Click on Close button	• Bookings form closes. • BookingGrid form still shows booking. • Booking still shown in Bookings table.
5	Click on free slot	• Check bookings still work OK. Repeat tests from Fig 4.4.5	• As in Fig 4.4.5.

Fig 4.5.2 possible test plan for deleting bookings.

4.6 Some improvements that could be made

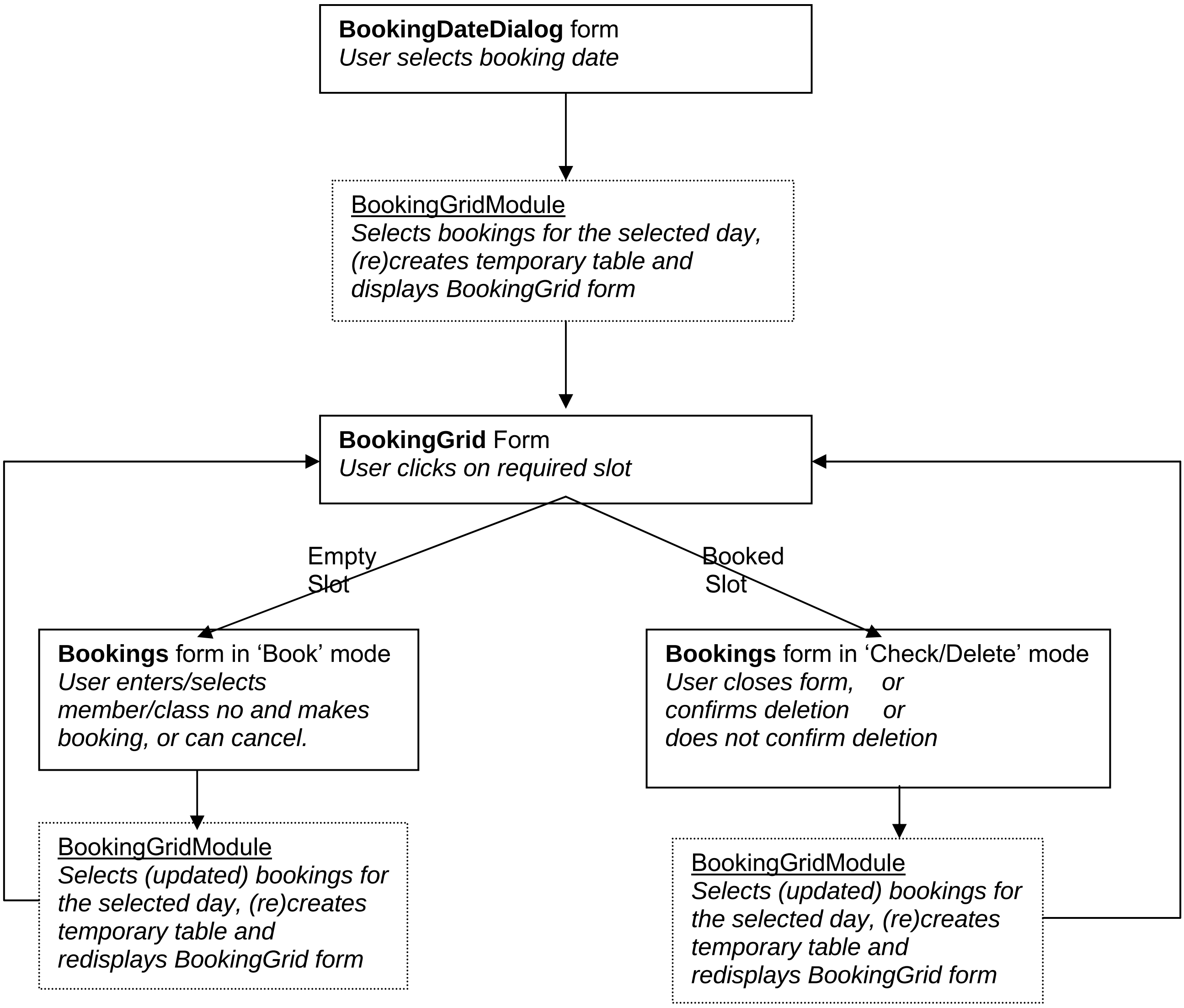
You have now seen how to use DAO code to create a diary page booking form which shows the Booking No for each booked slot, and how to use this form to make, check and delete bookings. The form is refreshed after each action, to show the latest bookings for the required date.

Item	Comments
Only book/delete for date in future, or today for time after now.	Cannot change (though can check) the past.
Confirmation messages ('Booking made OK' etc)	Useful information for the user.
Check for double bookings when putting values in array elements.	Will highlight errors that have already happened!
Each intCounter value should be checked to be in the correct range (0-11) when putting values in arrays.	If incorrect, will get run time error 'subscript out of range'.
When booking, check that user has entered a (valid) membership or class no. Using a list box may help.	Using a list box will also assist the user to select the correct member.
Show member name or class activity on booking form.	Useful information for the user.
Record attendance (easy to do via check/delete).	Needs extra field on Bookings table.
Use different headings on Bookings FVBA form according to task: e.g. "Make a Booking", "Check/Delete a Booking".	Useful information to user.
There needs to be a check to prevent double bookings.	This is discussed in Part 8 of the 'Getting Started' VBA Trainer.

Fig 4.6.1 Suggestions for improvements to the diary page bookings; work out your own test plan.

4.7 Diagram showing how all this Part fits together

There have seen three forms and a separate (re-usable) code module in the preceding sections. This diagram may help you understand what is going on and how it all fits together.


BookingDateDialog form
User selects booking date
BookingGridModule
Selects bookings for the selected day, (re)creates temporary table and displays BookingGrid form
BookingGrid Form
User clicks on required slot
Empty Slot
Booked Slot
Bookings form in 'Book' mode
User enters/selects member/class no and makes booking, or can cancel.
Bookings form in 'Check/Delete' mode
User closes form, or confirms deletion or does not confirm deletion
BookingGridModule
Selects (updated) bookings for the selected day, (re)creates temporary table and redisplays BookingGrid form
BookingGridModule
Selects (updated) bookings for the selected day, (re)creates temporary table and redisplays BookingGrid form

Note how the code in the BookingGridModule is reused to refresh the BookingGrid form, by picking up the data from the (updated) Bookings table.

4.8 Booking for fractions of an hour

The code in the previous sections assumes that all bookings are for hourly slots (09:00, 10:00, etc, for an hour) as this is the way that the Chelmer Leisure Centre requires bookings to be made. However, many appointment bookings (Advice Centre, Medical Centre, seeing lecturers) are normally made at smaller intervals, for example 09:00, 09:15, 09:30, etc.

The code in the previous sections could have used 9 to 20 as the array references (see Fig 4.3.1), as this range would have corresponded exactly to the hourly times being used. However, a range starting from 0 (zero) was used, so that the code could be adapted to allow for bookings of several slots in an hour.

Consider an organisation that requires bookings to be made at 15-minute time intervals, i.e. 4 bookings to each hour. The table in Fig 4.8.1 shows how the value could be calculated for intCounter.

Booking time	Multiplication factor (F)	Counter for the hour (hour x F)	Counter for the quarters	Final value of Counter
09:00	This is the number of bookings in an hour. For 15-minute bookings this value is 60/15 = 4.	09 - 09 = 0xF = 0		0
09:15			0 + 1 = 1	1
09.30			0 + 2 = 2	2
09:45			0 + 3 = 3	3
10:00		10 - 09 = 1xF = 4		4
10:15			4 + 1 = 5	5
10:30			4 + 2 = 6	6
10:45			4 + 3 = 7	7
11:00		11 - 09 = 2xF = 8		8
Etc…				

Fig 4.8.1 Determining the intCounter value

So, make the following changes to the BookingGridModule code:

- Fig 4.3.1 - change the lines for the following two declarations:
  - Const myconTimeInterval = 15 'minutes
  - Const myconEndTime = 44 '09:00 to 20:00 inclusive at 15-minute intervals
- Fig 4.3.2 - change the line for the DateAdd statement to add in minutes ("n" instead of "h")

```
dtBkgTime = DateAdd("n", myconTimeInterval, dtBkgTime)  ‘add 15 minutes to get next slot
```

- Fig 4.3.6 - after the line

```
intCounter = Hour(rstBookingTable.Fields("time")) - Hour(myconActualStartTime)
```

add:

```
intCounter = intCounter * 60 / myconTimeInterval       'for the hours
intCounter = intCounter + (Minute(rstBookingTable.Fields("Time")) / myconTimeInterval)  'for the minutes
```

- You will then need to restore the vertical scroll bars on the BookingGrid form as the form will now be much longer than before.

And that is all you need to do!

In fact, if you did the following initially:

```
Const myconEndTime = 11 * 60 / myconTimeInterval
```

in Fig 4.3.1

and made all the other changes listed above, then you would only need to change myconTimeInterval to the required number of minutes [60 (for an hour) or 15, or other intervals that divide exactly into one hour] and the rest would work unchanged.

Try the code out and see the result. Make, check and delete bookings. (Careful - if you want to change back to hourly bookings, delete the bookings at 15, 30, 45 minute timings first or restore from a backup of your Bookings table).

Finally - note that this code does not cater for the case where bookings are not continuous throughout the day. A medical practice, for example, may have bookings only at certain times, e.g. 08:30 to 12:00 then 16:00 to 18:00. It is up to you to work out how to adapt the above for this situation. One way could be to have a table of allowable booking times, and check (perhaps using DLookup) the calculated time against this table when writing the array rows out.

4.9 Exercises

Implement the improvements suggested in section 4.6.

END OF MAIN PART OF TRAINER

The Index follows on the next pages.

INDEX

Item **Section**

#, for date and time 4.3.1, 4.3.3
&, to concatenate items 1.4.3, 3.3, 3.4, 4.3

A
acDataErrAdded 3.3
acDataErrContinue constant 2.4.1, 3,3
acDataErrDisplay constant 2.4.1
acForm constant 2.4.2
acFormAdd constant 4.4.1
acTable constant 4.3.5, 4.3.7
AddNew method 3.2.4, 3.2.5, 3.3, 4.3.5
Admin, default Access user name 1.3
ADO code See DAO code
AfterUpdate event, text box field 1.4.3, 4.2
Aggregate functions in SQL 3.4.2, 3.4.3
AllowAdditions property 4.3.6
Application object 1.3
ApplyFilter method 1.4.3, 1.5
Arrays 4.3
Asc function 1.6.2

B
BackStyle property 4.5
BeforeUpdate event, text box field 2.3.3
BOF property 3.2.4
Bookmark property 3.2.4

C
Cancel parameter 4.2
Case sensitive check See Password
Case statement 2.3.3, 4.3.4, 4.3.5
Chr function 1.5
Click event for command button 3.4, 4.2, 4.4.2, 4.5
Click event for text box on form 4.4.1
Close method 1.4.3, 1.5, 3.2.4, 3.2.5, 3.3, 3.4, 4.3.5, 4.4.2, 4.5
Combo box 3.3
Conditional formatting 4.3.6
Const 2.3.1, 3.2.5, 3.4.2, 4.3.1, 4.8
COUNT, SQL aggregate function 3.4.2
CREATE TABLE SQL 3.2.2, 3.2.5, 4.3.5
CurrentDB method 3.2.1, 3.3, 3.4, 4.5
CurrentUser method 1.3

D
DAO code Part 3, Part 4
DAO Object Library 3.1.4
DAO.Recordset See Recordset
Data Access Objects See DAO code
Database object 3.2.1, 3.3, 3.4, 3.6, 4.3, 4.5
DataErr parameter 2.4.1
DateAdd function 4.3.2
DateValue function 4.3.3
DCount function 3.3
Debugger 4.3.2, 4.3.4
Delete a table See DeleteObject
Delete method 3.2.4
DELETE SQL statement 4.5
DeleteObject method 4.3.5, 4.3.7
Dim statement 3.2.1, 3.2.4, 3.2.5, 3.3, 3.4, 4.3.1
DLookup function 1.5, 3.4
DO loop 3.2.5
DROP TABLE SQL 3.2.2, 3.2.5

E
Edit method 3.2.4
Enabled property 4.2
Encryption 1.4.5
EOF property 3.2.4, 4.3.4
Err object 2.1, 2.3
Error codes 2.2
Error event in form or report 2.2, 2.4
Error message 3211, database engine couldn't lock table… 3.6
Error message, no current record 3.2.5
Error message, object variable or with block variable not set 3.2.2
Error message, sub or function not defined 2.1, 4.2
Error message, table already exists 3.2.2
Error message, user-defined type not defined 3.1.4
Error message, variable not defined 3.2.1
Error, break on all 2.2
Execute method 3.2.2, 3.2.5, 4.3.4, 4.5
Exit Sub 2.1, 4.4.2
External database 1.4.1, 3.2.5

F
Fields property 3.2.4, 3.2.5, 3.3, 3.4, 4.3.4, 4.3.5, 4.8
For loop 1.6.2, 4.3.2, 4.3.5
Format property 4.3.6
FormatDateTime function 4.2, 4.3.6
Forms Collection 1.4.3, 4.3.6, 4.4.2, 4.5

H
Hour function 4.3.4, 4.8

I
If statement 1.4.3, 1.5, 2.3, 2.4.1, 3.3, 3.4, 4.3.4, 4.4.2, 4.5
INSERT INTO SQL 3.2.2, 3.2.5
InStr function 1.5
IsDate function 4.2
IsMissing function 3.4
IsNull 1.4.3, 1.5, 4.2, 4.4.1

L
Labels in code 2.1
Left function 1.5
Len function 1.6.2
Linking an external table 1.4.1
Load event, form 1.4.2, 4.3.6, 4.4.2
Locked property 4.5
Login form 1.4.2

INDEX

Item **Section**

M
Mid function 1.6.2
Minute function 4.8
MoveFirst method 3.2.4, 3.2.5, 4.3.4
MoveLast method 3.2.4, 3.2.5, 3.4, 4.3.3
MoveNext method 3.2.4, 3.2.5, 4.3.4
MovePrevious method 3.2.4
MsgBox function 2.1, 2.3, 3.2.5, 3.4, 4.3.3, 4.4.2
myCheckPassword function 1.6.2
myCheckSlot procedure 4.4.1
myDCount function 3.4.1, 3.4.2
myDGetData function 3.4.4
myDisplayBookingGrid procedure 4.3, 4.4.2, 4.7
myDisplayInfoMessage procedure 1.4.3, 1.5, 4.3.4
myDisplayWarningMessage procedure 1.4.3, 1.5, 2.3, 4.2, 4.3.4, 4.3.5
myDLookup function 3.4.3
myGetCurrentUser function 1.5
myHiddenFields procedure 4.4.1
myYesNoQuestion procedure 3.3, 4.4.2, 4.5

N
Network login 1.5
NoMatch property 3.2.4
Nothing keyword 3.2.5
NotInList event, combo box 3.3
Null constant 4.3.2

O
Object Browser, the 3.5
On Error clause in procedure 2.2, 2.3, 4.3.4, 4.4.2
OpenForm method 1.4.3, 1.5, 4.3.6, 4.4.1, 4.5
OpenQuery method 2.1
OpenRecordset method 3.2.4, 3.2.5, 3.3, 3.4, 3.6, 4.3.3, 4.3.5
Optional parameter 3.4

P
Password, Access simple 1.2
Password, case sensitive 1.6.2
Password, input mask 1.4.2, 1.4.4, 1.6.2
Public 2.3.1, 3.2.5, 4.3

Q
Quit method 1.4.3, 1.5

R
RecordCount property 3.2.4, 3.2.5, 3.4, 4.3.3, 4.3.4
Recordset Object 3.1.4, 3.2.4, 3.2.5, 3.3, 3.4, 3.6, 4.3
Requery method 3.2.4, 3.2.5
Required property 2.3.3
Response parameter 2.4.1, 3.3
Resume statement 2.1, 2.3.1, 2.3, 4.3.5, 4.4.2

Item **Section**

S
Security, Access user level 1.3
Seek method 3.2.4
SELECT SQL 3.2.4, 3.4, 4.3.3
Set statement 3.2, 3.3, 3.4
SetFocus method 4.3.6, 4.4.2, 4.5
SpecialEffect property 4.5
Startup 1.4.3

T
Test plans 1.4.4, 4.3.3, 4.4.2, 4.5

U
Undo method 4.4.2
Update method 3.2.4, 3.2.5, 3.3, 4.3.5

V
vbCrLf 2.3, 3.3
vbLongDate constant 4.2, 4.3.6
vbNo 3.3
Visible property 1.4.3, 1.5, 4.4.2, 4.5

W
Weekday function 4.3.6
WeekdayName function 4.3.6
With statement 3.3, 4.3.5, 4.5