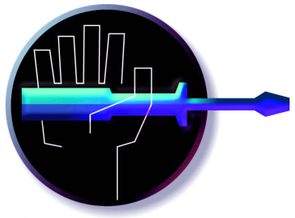


A sporting chance

Visual Basic gives the mythical *PCW* sports club the database treatment in the second of our workshops, conducted by Tim Anderson. He also studies the life of a VB object.

Last month's workshop showed how to create a simple database application for the *PCW* sports club. It was a flat-file database, which means all the data was stored in a single table, like a card-index. At the sports club, though, it is important to know which sports a member has signed up for. A member can sign up for any number of sports, and each sport is played by any number of members. This is a classic many-to-many relationship, but it is not always obvious how best to store this kind of information.

One strategy would be to add several fields to the table of members, for Sport1, Sport2, Sport3, etc. Another possibility is a notes field with the sports entered line by line. Both these ideas are fatally flawed. Although they seem easy, they are actually inefficient and inflexible. For example, what happens when you want a list of all the footballers? You would end up with a horrible keyword search and probably get inaccurate results.

The correct approach is to analyse the data into three tables. The first one is the table of members. Next there is a table of sports, which for the moment has just two fields, Sport and ID. The third table records which member belongs to which sport. SportLink again has two fields, MemberID and SportID. If you view the table on its own it will look like a meaningless string of numbers, but in combination with the other tables it makes sense. Later you might want to add other fields to SportLink, perhaps a Role field which contains information like "Goalkeeper" or "Captain". It is important to grasp this principle, which is a great way to store all kinds of data.

The main form needs adapting to display this new information. Since a member may sign up for any number of sports, these are best displayed in a listbox control. To keep the form from getting cluttered, a good tip is to use a tab control as well. Visual Basic 4.0 comes with two: a TabStrip which is for Windows 95 only, and the SSTab which comes as both a 16-bit and 32-bit OCX. In this example SSTab is used. The tabs work at design time, making it easy to lay out the form. When a tab is selected, controls placed on it belong to that tab. Controls placed on the form itself will show through all the tabs. Fig 1 shows buttons for adding and removing members from particular sports, but these are not yet enabled.

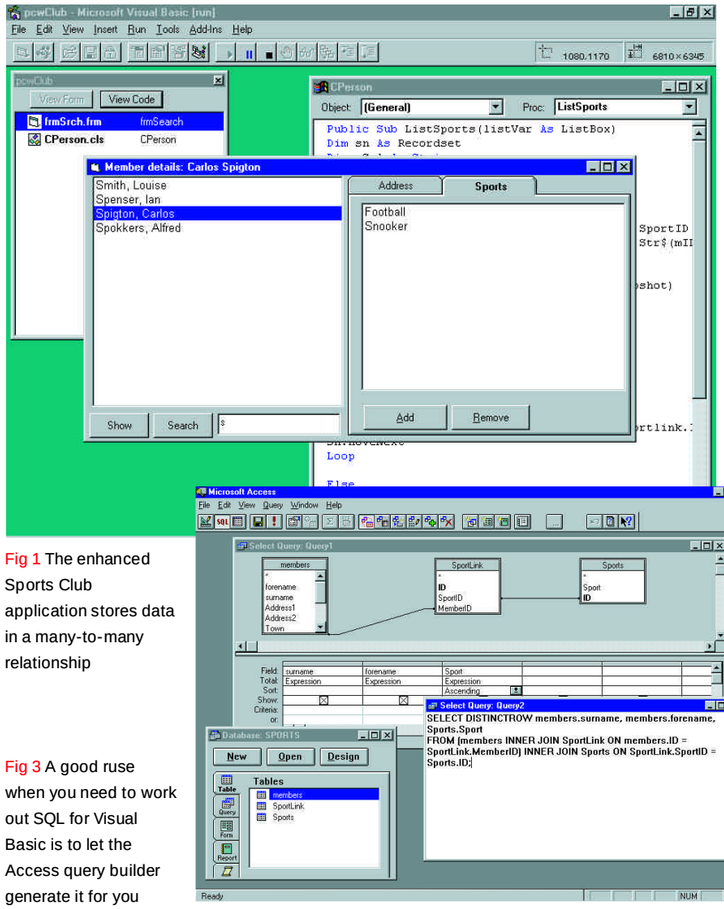


Fig 1 The enhanced Sports Club application stores data in a many-to-many relationship

Fig 3 A good ruse when you need to work out SQL for Visual Basic is to let the Access query builder generate it for you

The next step is to write code to display the list of sports for each member. One idea is to add a ListSports method to the CPerson class. The ListSports method takes a listbox control as a parameter. It searches the database to get the list of sports and adds them to the listbox. Doing it this way means that if a list of sports is needed at some other point in the application, it will not be necessary to rewrite the code. All you need do is to supply the ListSports method with an available listbox. The code for CPerson.ListSports is in Fig 2.

Much of this code is similar to that used last month for searching the members table. The main difference is that the SQL query for extracting data from two tables is more complex. If you followed *PCW*'s recent SQL workshop, you will have no problem. If not, notice that the SQL string includes several sections:

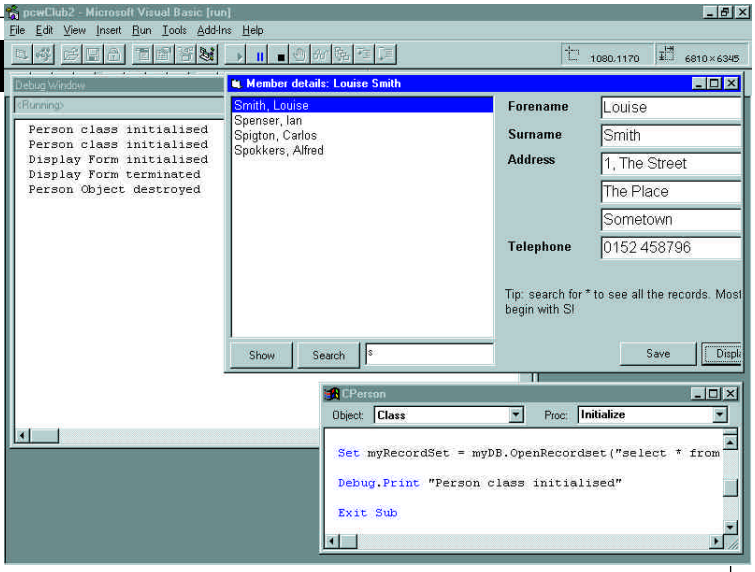
1. Which fields to extract — SELECT * for all fields.
2. How the two tables are linked — the first part of the WHERE clause.

p268 ➢

Fig 2 The code for CPerson.ListSports

```
Public Sub ListSports(listVar As
ListBox)
Dim sn As Recordset
Dim sSql As String

listVar.Clear

' build up the SQL command
sSql = "SELECT * FROM Sports,
Sportlink "
sSql = sSql & " WHERE Sports.ID =
Sportlink.SportID "
sSql = sSql & "AND
Sportlink.MemberID = " & Str$(mID)
sSql = sSql & " ORDER BY
Sports.Sport"

Set sn = myDB.OpenRecordset(sSql,
dbOpenSnapshot)

' now fill the list box
If Not (sn.BOF And sn.EOF) Then
' there are records

sn.MoveFirst

Do While Not sn.EOF
listVar.AddItem (sn!SPORT)
listVar.ItemData(listVar.NewIndex)
 = sn![SPORTLINK.ID]
' the square brackets and table
 name are needed because
' there are two different ID fields
\ in the result set
sn.MoveNext
Loop

Else

listVar.AddItem "None"

End If    End Sub
```

The life of a VB object



Judicious use of Debug.Print can help track the lifetime of VB objects

Introduced in Visual Basic 4.0, class modules are a way to create user-defined objects. For example, the Sports Club application has a CPerson class with properties and methods. These constitute the interface which a person object presents to the application. Whenever your other code has to interact with a person object, it does so through this interface. If the interface stays the same, you can change or improve its implementation (the code which drives those properties and methods) with no danger of breaking the application. If you add to the interface, those new features are available wherever a Person object is referenced.

The first thing to understand is the lifetime of an object. Unlike other variables, you can't simply DIM a CPerson object and then refer to its properties. Objects must be instantiated. For example, this gives an error:

```
Dim Myperson as CPerson
Myperson.surname = "Baxter"
```

Error — object variable not set. Instead you need code like this:

```
Dim Myperson as Cperson
set Myperson = New CPerson
Myperson.surname = "Baxter"
```

Or if you start with:

```
Dim Myperson as New Cperson
```

VB will instantiate the object when first referenced.

The question of instantiation is important because it does not just allow you to start using the object. It fires an event, Initialize. All class modules have this event predefined. It is extremely valuable, since you can do things like setting default values for properties, or opening a link to a database, or instantiating other subsidiary objects as required. Sadly, Initialize cannot take parameters, making it less useful than it should be. There is another, similar event called Terminate, which occurs when the object is destroyed.

But when is the object destroyed? It is destroyed when there is no longer any active reference to it in your code. This feature is designed to make it easy to manage objects, but can get confusing. If you have an object variable declared in a procedure, it goes out of scope and the object is destroyed when the procedure finishes. But if you have assigned the object to another variable which is still in scope, the object is not destroyed: the listing (left) illustrates the point. So far, not too difficult. It's harder when your objects are more ambitious. Perhaps you want a CPerson to have a Display method which creates and shows a form. VB forms are just another kind of class, and the obvious approach would be like this:

```
public sub Display()
Dim myform As New DisplayForm
' ... code to fill the fields
myform.Show
```

But to encapsulate things you will want a Hide method which disposes of the form. That means keeping a reference to the form in the CPerson class, so the DisplayForm variable needs to be scoped to the class. It is likely the form will need to interact with its corresponding CPerson object, so you give the form a Person property. The references start to proliferate, and neither the DisplayForm nor the Person object will be destroyed until the last one goes out of scope or is set to Nothing. If your Hide method was like this:

```
Unload myform
```

that would not destroy the form object. In turn, the form object would prevent the Person object from being destroyed, because it still has an active reference to it. You have to add the line:

```
Set myform = Nothing
```

to clean it up properly. The conclusion is that you need to watch the lifetime of VB objects closely or they could stick around for longer than they are wanted.

Destroy all objects!

```
Dim Myperson As New CPerson
Myperson.surname = "Baxter"
Set FormPerson = Myperson
' assumes FormPerson is scoped to the form
' both now refer to the same object
Set Myperson = Nothing ' Object is NOT destroyed
Set FormPerson = Nothing ' Object is destroyed
```

3. An additional restriction — the second part of the WHERE clause after AND. This ensures that only data for the current member is extracted.
4. An ORDER BY clause to sort the results. Working out SQL acceptable to JET, the VB database engine, can be tricky. A good ploy is to build a query in Access, then cut-and-paste the generated SQL code (Fig 3). Note that often, more than one SQL expression will product the same result, sometimes with performance differences.

■ *All the code for this month's VB workshop is on this month's cover CD.*

■ *Next month: Visual Basic, inheritance and delegation.*

PCW Contact

Tim Anderson welcomes your comments and queries. Write to the usual *PCW* address, or email freer@cix.co.uk