And that's magic...

Mark Whitehorn conjures up tricks to use in Access. He finds it is a product that knows where its towel is, without any flannel, and has meta-data which needs to be stored.

You can use Shift F2 virtually anywhere in Access. It opens a "Zoom box" where you can edit the text upon which your edit cursor was sitting. Use it whenever the text you are trying to read sits in a small box. It is mega-useful when entering a complex condition in a query, say. It works for users of your application who can be encouraged to use Shift F2 when entering data into a form.

You can also use Shift F2 while programming. If you place the edit cursor on a user-defined function, pressing the magic key combination will jump you to the definition of that function. Doing the same on an Access function like CurrentDB (i.e. not user-defined) will open up the object browser at the appropriate place.

Committing records

Another useful tip to feed to your users is that Shift Enter will commit an edited record to disk without the need to move onto the next record. (Or click on the edit record symbol on the left of the form.)

Fig 1 **Inside the data dictionary. Intuitive, isn't it?**

Access currency formats

If you choose currency as data-type for a field, Access will use whatever Windows has been told is the default currency (via the control centre). However, once a data-type has been assigned to be dollars, say, Access won't change this to pounds if the data file is moved to a Windows machine where the default currency is pounds. This makes perfect sense (since the absolute value of items would alter as data files were moved from country to country) but needs to be remembered if you are distributing an application into other countries.

A free toy for data mining in the data dictionary

Access is a product that really knows where its towel is. Well, in the literal sense it has no idea of the physical location of its textile de-humidifying implement; in fact, it probably hasn't got one. What it does have is the database equivalent, which is called meta-data, and it certainly knows where that is — in the data dictionary. Your problem is trying to read that data and I have a solution.

A database consists of the user's data stored in tables. A DBMS also has to store information which "describes" the database itself (the names of the tables, the names of the queries, the explicit joins between the tables, the data integrity constraints etc). Such information is known as meta-data. One of Codd's rules that define a Relational DBMS says that the meta-data can't be stored any old way; it must be stored as data in tables, just like the user's data. Access obeys this rule and stores the meta-data in a series of hidden tables, the names of which all begin with MSys (presumably standing for Microsoft System). The data in the tables is the meta-data, the tables themselves can be referred to as the data dictionary.

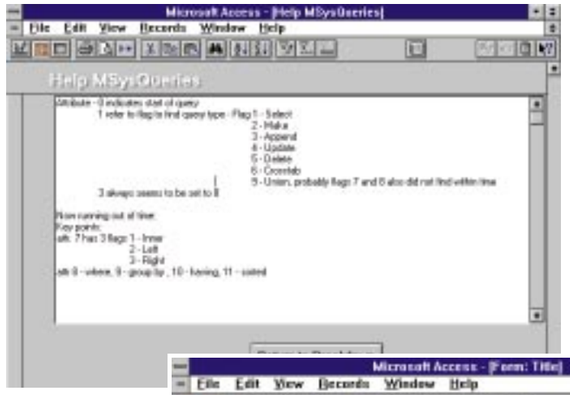
Codd says that storing the meta-data in the same way as the user data ensures that the users of the database can access the meta-data in the same way in which they access their normal data. So any user who has learnt to query their data to find out, say, how many customers live in Hereford, can use the same techniques to find out how many forms there are in the database.

All of this is fine and dandy in theory but in practice it rarely works, for a very simple reason: the database designers are free to use whatever data format they like within the tables to represent the structure of the database, and they seem to choose very abstract formats. In Access, for example, data about the queries is stored in a table called MsysQueries (Fig 1).

The first 15 rows in the table shown, define one query in the database. In order to find out what sort of query it is, you have to find the value 1 in the Attribute field within the first 15 rows and then look up the value in the field called Flag.

Flag value	Meaning
1	Select
2	Make
3	Append
4	Update
5	Delete
6	Crosstab
9	Union

p266 ➢



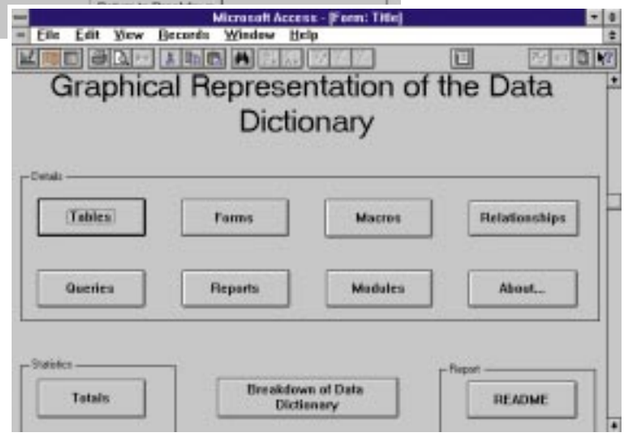


Fig 2 (left) Some of the useful information inside Black.mdb

Fig 3 (below) The GUI front-end to Black.mdb

I don't know about you, but I couldn't have worked that out from a quick glance. None of this is documented (at least for normal mortals) so how did I find out? For my sins, I teach the database course at Dundee University. One of the projects I set the students this year was to build an Access application that displayed the meta-data in a user-friendly fashion. In order to do this, the students had to "reverse engineer" the data they observed in the data dictionary, working out what all the bizarre formats meant before generating queries that made sense of them.

Student demonstration

It seemed a shame for this work to be wasted, so I am including one of the better ones on our cover-mounted CD-ROM (in Black.mdb). This was written by one Johnny Black and includes some notes he made about the formats used within the data dictionary.

Three riders: first, remember this was a student project, so neither Johnny nor I can guarantee that all his conclusions are correct. Second, the structure of the data dictionary varies between different versions of Access. Black.mdb was written specifically for version 2.0 and may well not provide accurate information if imported into later versions. Third, this work was completed to a tight time schedule and had I given the students more time could obviously have been improved. With those riders, Black.mdb should still serve as an excellent starting point for anyone who wants to examine the data dictionary.

Operating theatre

Having covered the hardware aspect of client–server computing, we can move on to a more contentious area: the operating system and the back-end DBMS.

Assuming for a moment that you do not want to stray into the realms of mainframes just yet, the obvious operating systems onto which a PC database application could be up-sized are UNIX, Novell NetWare and Windows NT.

UNIX is possible, but my experience is that most PC users tend to shy away from it, which is a shame because it's a very stable platform for RDBMSs.

Next comes NetWare, which some people may consider, especially if they are using it as their NOS for file and print. The problem is that NetWare was optimised for precisely those functions, not as an application server. So although products like Oracle 7 are available for NetWare, in practice they sell like cold cakes.

I love NetWare dearly as a NOS but cannot recommend it as an operating system for RDBMSs.

Finally there is Windows NT. This is not as stable as UNIX, and we could debate for some time whether it is really a stable enough platform for a mission-critical RDBMS. It is apparent however that many people are migrating to it and that the major RDBMS firms are providing products for it. If you choose to use Windows NT, at least you will have lots of company, so that's my general recommendation unless you have good reason to choose otherwise.

Each of the big three has an offering for NT: IBM has DB2 for NT, Oracle has Oracle for NT, and Microsoft has SQL Server. Two of these products have a history of running on other platforms, and all three have perceived strengths and weaknesses.

DB2 is famous for running on mainframes, being very big, powerful and secure, and being a pig to drive. Oracle is famous for running on UNIX, being very big, powerful and secure, and being a pig to drive. SQL Server is famous for running on NT and having a great user interface.

If you are currently an Access user, the obvious choice is surely SQL Server from Microsoft. After all, you are already happy with Access: SQL Server comes from the same company so you can expect the same quality of software. You can also expect a high level of compatibility between the front-end and the back, so is there even a question here?

The answer is "Yes", there is a serious question, but a relatively complex one with several facets. Access and SQL Server were both developed within Microsoft but by different development teams. My experience is that the teams can produce differing standards of products, so the fact that both of these come from Microsoft is largely irrelevant. Indeed, SQL Server was originally a Sybase product and still has large chunks of Sybase code buried within it which has no connection with Microsoft at all. This code includes the query-optimiser which is "less-than-optimal".

In my June issue column I suggested that many client-server databases are also mission-critical to a greater or lesser extent. There is a huge gulf between a word processor and a database which stores information crucial to the survival of a company. It generally doesn't matter much if the former falls over occasionally: the latter must be much more stable and, if it fails, must fail gracefully so it can continue to work when restarted.

Companies like Oracle and IBM have always worked in mission-critical environments and understand that, in this environment, the stability and data-integrity of the product is of paramount importance. They understand that software mustn't be shipped until it is really rock-solid.

Now ask yourself what you think about Microsoft's track record in producing software that never falls over? Ask yourself how many times you have heard about bugs turning up in new releases of Microsoft products which will "be fixed in the next version". I'm not suggesting that Microsoft can't learn about mission-critical software. What I am suggesting is that I want to see a good track record before I commit my data to a given product. I believe IBM and Oracle have already demonstrated such a track record.

"Ah, but what about drive-ability?" you ask. "You're really suggesting that we use a product with a Neolithic user-interface." (Those of you who haven't experienced the joy of playing with the current version of DB2 for NT may be surprised to discover that it still has a command-line interface. No, I'm not joking, or exaggerating. Yes, I do mean you don't get cute little buttons, icons and slider bars. You have to type a string of characters into a textbox in order to drive it.)

But no, I wouldn't inflict a command-line on anyone. The good news is that I have been playing with the new version of DB2 Universal Database for NT (due for release in October) which has a really intuitive, easy-to-use interface. In fact, it closely resembles Access (Fig 4).

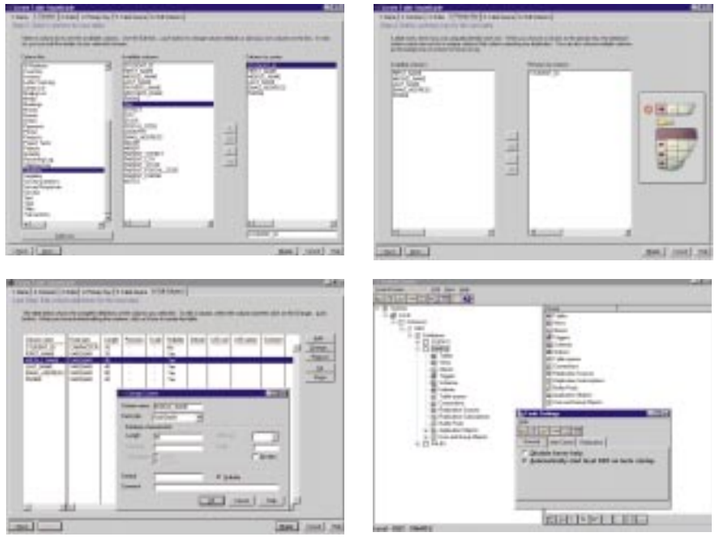


Fig 4 Using DB2 for NT is actually a pleasant experience, at least in the new version. Clockwise (from top left): Choosing fields for a table; setting the primary key; setting data types; the main control area for DB2

Of course, if we are allowed to talk about next versions, Microsoft is likely to say the next version of SQL Server will fix any current problems that exist within the product. Apart from considerations about track records in this area, a good question to ask here is, "Which company has the easiest job?" IBM, which has to bolt on a respectable interface to an already mission-critical piece of software, or Microsoft, which has to convince us that SQL Server is of mission-critical quality.

Scaling up

Scalability is loosely defined as: "How much the performance of a given product alters as conditions are changed."

A good example is the number of users attached to an RDBMS. Suppose you have an RDBMS running with 20 concurrent users. If it were perfectly "scalable", doubling the number of users would no more than double the response time for each given user. But life isn't usually like that, and doubling the number of users often makes the response time more than double for each user (because the RDBMS spends proportionately more time swapping between the users). So no-one expects perfect scalability but it is possible to compare products and see which scale most efficiently. Bloor Research has recently published a report called *The Realities of Scalability* which examines the scalability of DB2 for NT and SQL Server (it also looks at the scalability of DB2 for AIX).

The scalability of SQL Server comes in for criticism: "Put simply, Microsoft SQL Server for Windows NT at high numbers of users performs dramatically worse than either of the other two databases." As for the stability: "This database had a number of failure states that could repeatably [*sic*] be generated. With large numbers of users, it was found that the database would grind to a halt."

DB2 for NT, as you might expect given its history, does not come in for such criticism: "Despite the occasional unexplained server or database crash, it proved impossible to consistently generate a fatal error in DB2 for Windows NT. It seemed to be capable of taking most things thrown at it."

Perspective

All of this has to be seen in perspective. Suppose you work for a small to medium-sized enterprise. You have developed an Access database on a standalone PC and it has been riotously successful. Now your boss wants you to upgrade it so that ten people can use it. There is no suggestion, either from the Bloor report, or from me, that SQL Server would be inappropriate. However, I can see no reason why the new DB2 wouldn't be better and it is actually cheaper, so it is well worth considering.

Now suppose you are developing an application that may become mission-critical in the future, or which needs to support many users (say >1,000) and/or lots of data (say >100Gb). Under these circumstances, for the reasons given above, I would strongly recommend that you look carefully at the alternatives to SQL Server rather than view it as the default option.

PCW Contacts

Mark Whitehorn welcomes readers' correspondence and ideas for the Database column. Write to him at the usual *PCW* address or email database@pcw.vnu.co.uk.

Bloor Research 01908 373311

Classy chassis

Mark Whitehorn turns his attention to "professional" quality hardware for client server databases and what you can expect for your money. Plus, tips and tricks for Access.

Last month I was trying to "sell" you a very expensive piece of kit. I hoped to convince you that standard PC chassis were inappropriate as servers for mission-critical databases and that you ought to buy something expensive from a reputable manufacturer.

A fair question here is, "What do I get for my money?" In other words, a "professional" server is likely to cost twice as much as a PC of the same specification, so where does the money go? The following list is not exhaustive but it gives some idea of the extra hardware features you can expect from a "professional" server:

- RAID disk arrays.
- Fault monitoring and prediction on hard drives, processors and memory (anyone who remembers HAL at work in 2001 — A Space Odyssey will appreciate this).
- Pre-failure warranty, which allows the components identified as liable to failure, to be replaced before they do so.
- Dual peer PCI buses for high I/O bandwidth.
- Array controllers which allow RAID storage to be added while the system is in operation.
- Tape backup systems which provide rapid backup (40Gb per hour).
- Redundant power supply and UPS options.
- Redundant controllers on standby in case of failure.

Most of these are self explanatory. However, a brief word about RAID (Redundant Arrays of Inexpensive Disks) may be helpful. "Brief" is the operative word, as there are several flavours and the area is relatively complex. A RAID allows you to store your data on several disks, which holds a number of advantages as far as database servers are concerned. For a start, a RAID can be set up in such a way that if even one of the disks fails, your data remains safe because each piece is stored redundantly on more than one disk.

Clearly, this has the disadvantage that although you might have five 2Gb disks and therefore 10Gb of disk space, you won't be able to store 10Gb of data. The data is more secure, though. Additionally, there is a rather elegant spin-off from the redundant nature of RAID which is that the data can often be retrieved more rapidly.

Furthermore, RAID can be hot-swappable, so if a disk fails you can extract the duff one, slot in a good one and the data which was on the failed disk will be reconstructed from the others. The database can continue to perform, albeit more slowly, until the new disk is fully online.

It doesn't take a genius to work out that the money is going into two main areas: speed, and keeping the server up and running. You may never need the RAID array; that redundant controller may spend all its time idling. Nevertheless, it would make your day having arrived one morning to discover that even though one of the disks has failed, your system is still running.

Remember it well

A database server needs RAM, and then it needs more RAM. A database server will use RAM to hold the RDBMS engine and to cache data, transactions and indices. The more RAM you give it (within reason) the faster it will work. Think in terms of 64Mb as a minimum and 100Mb as a more reasonable starting point. Make sure the server you buy has free memory slots so the

Time, gentlemen, please…

Following reader Gareth Wade's query about adding time lengths to give a running total, (*Quickies, PCW* April) I have since received a couple of solutions from others.

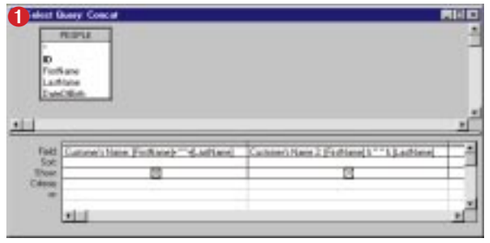
■ MA Roberts writes: "*Regarding the 'Database Quickie' from Gareth Wade (PCW April) I have had a similar problem to Gareth's when adding together time values. The short time format field in Access does not work satisfactorily for adding values that will exceed 23:59. One workaround I am using is to convert the time field into a number type. When adding time records together in a number-type field, Access converts the time to a fraction of the 24-hour day. It is therefore necessary to multiply this field by 24 to get a value for hours and decimal minutes. This field may then be used to obtain a value for time (in hours and decimal minutes) multiplied by an hourly rate such as might be used in a time-sheet style application.*"

■ And Vidar Eggen writes from Oslo: "*I had the same problem when wanting to report on the added flying time of pilots as well as aircraft at my flying club. What we really need in Access is the [ttt:mm] format of Excel. Until then, I use this basic formula:*

```
=Str$(Sum(Hour([Time]))+Sum(Minute([Time]))\60) & ":" &
Format$(Str$((Sum(Minute([Time]))) Mod 60);"00")
```

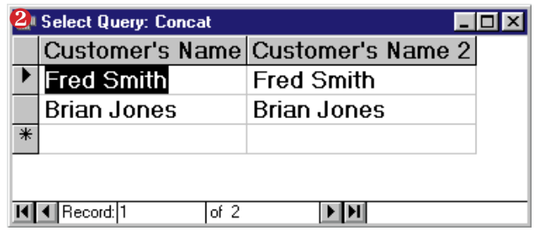
"*The output of this formula will be '123:45' in text format. The reason for converting the sums to a string is to make sure that five minutes is printed as '05', not '5'. If quoting from this, please rephrase my language into readable English!*"
(Vidar's English seems fine to me and is considerably better than my Norwegian! — *MW*)

p284 ➢



machine and turned it loose on a large database. Rather to my surprise Access used all the four processors on the machine and was able to query 1,000,000 indexed records in a couple of seconds. Interestingly, Excel used only one of the available processors and its performance was startlingly unimpressive.

So the bottom line is, what do I use as my test database server? I am currently running a Compaq ProLiant 5000 with four 166MHz Pentiums, 384Mb of RAM and 9Gb of disk space. I look upon it not so much as a server, more as a non-intrusive sleeping aid. Okay, it is more expensive than the tablets, but it has fewer side effects.

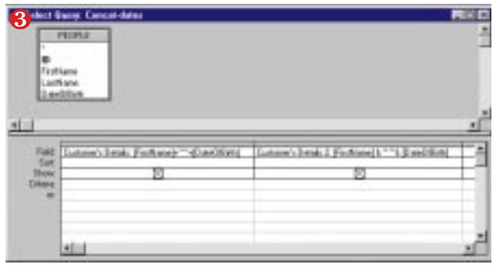


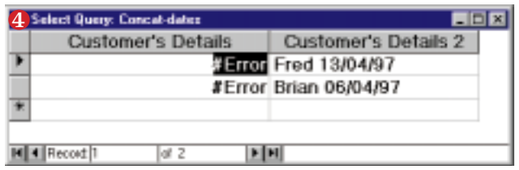
memory can be expanded later without replacing all the RAM it currently holds.

Purchasing processors

In general, buy several processors. Simple operating systems can only use a single processor, but complex ones like UNIX, VMS and NT4 Server can use multiple processors. Mind you, just because an operating system can make use of several CPUs doesn't mean that the programs which run on top of it can too. But all the respectable RDBMSs can and do, and this isn't restricted to client-server RDBMSs.

I recently mounted Access 97 on an NT4





Listing 1

```
"Report generated " & Now()  & " for " & [Customer]
```

Listing 2

```
Between Date()-7
And Date()
```

Listing 3

```
Between #1/1/1981# And Date()
```

Tips and tricks for Access

By popular demand, I am introducing tips and tricks this month. It is aimed at Access users and will include information which ranges from simple to complex and should provide something for everyone.

1. **Queries**, by default, use the field name as the column name in the answer table. If you ever want to alter this, insert the text you would prefer to use, followed by a colon, before the field name. Thus:

```
Customer's Name: [FirstName]
```

will replace "FirstName" with "Customer's Name" in the answer table. Which leads us neatly to the second tip.

2. **The "+" operator** can be used to concatenate text strings, as many people know. Thus:

```
[FirstName]+[LastName]
```

in a query will join the two strings together. But this unhelpfully yields names like "BrianJones". So, simply add in a literal space like this:

```
[FirstName] + " " + [LastName]
```

which then gives: "Brian Jones".

However, you can also consider using the lesser known "&" operator. Amazingly,

```
[FirstName] & " " & [LastName]
```

gives exactly the same result: "Brian Jones" (Figs 1 & 2). You're thinking, "Hello, he's flipped. If it gives the same result, why bother to use it?" Well, the advantage of getting into the habit of using "&" is that it automatically converts all data types into text. This has no effect if the two fields are already text (as in the case above) but it does wonders with expressions like

```
[FirstName] & " " & [DateOfBirth]
```

which will produce "Brian 06/04/1967" rather than the "#Error" produced by the + operator when it tries to concatenate data from two disparate data types (Figs 3 & 4). In other words, if you habitually use "&" instead of "+" you won't have to use type conversion functions.You could even use it in a report like Listing 1.

3. **When adding fields to a query grid**, you can drag and drop them as usual. However, you can also select multiple contiguous fields using shift-click on the first and shift-click on the last, and then dragging and dropping as usual. You can also add non-contiguous fields by CTRL-clicking on them.

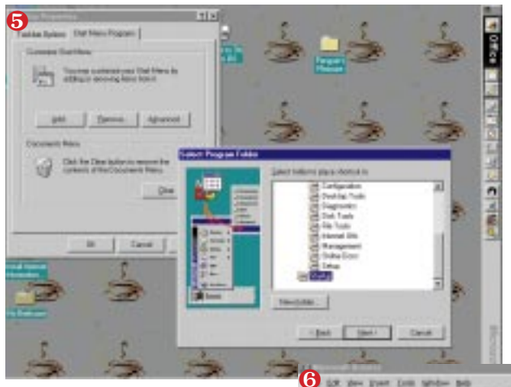
4. **When looking at a table of data**, you can resize any field to fit the widest data in the field by double-clicking on the right-hand column border (this has to be done at the top of the table). You can hide the column by dragging the right-hand column border across to the left-hand border. You *can* open this up again with the mouse but it takes practice, as you often simply widen the adjacent column instead. It is often easier to use the menu system Format

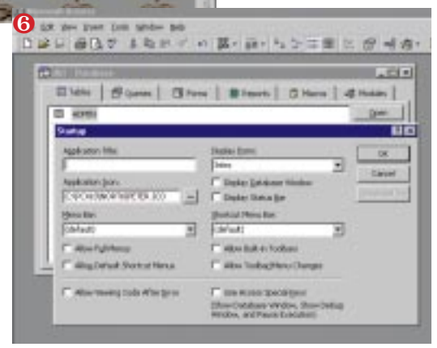
Listing 4

```
Set button OnClick event to "=Clicked([Screen].[ActiveControl].[Caption])"
```

Listing 5

```
Function Clicked(capStr as string)
    Debug.Print capStr
'Or assign it to the parameter for the query
End Function
```





Show Columns, or Format Unhide Columns, in Access 97.

5. **Record selector.** Most of us have discovered the delights of the "Find Specified Text" button (the binoculars) which will search a table for information. By default, this will search the current field and the dialog box allows you to specify the entire record. However, it is often easier to click the record selector (at the left of the record) before starting the search and this automatically sets the search to run against all fields. This works in form views as well as in datasheet views.

6. **"Between… And".** As regular readers will know, I am obsessed with the idea that the data in our databases should be as "clean" as possible. Take something as simple as a DateOfBirth field in a maternity database. The default (for the child, *not* the mother) might be the function Date().

But perhaps not every child's record might be entered on the day (staffing levels being what they are) so the simple answer is to allow any entry. But that opens the door to really whacky entries. Suppose we know that all entries, no matter how delayed, are always brought up to date on a Sunday, say. This means that the longest delay would be seven days, so the default could remain as Date() and the validation rule could be set as Listing 2. The "Between… And" operator is a useful one to get to know. You can use it with fixed dates like Listing 3 or for data types other than dates:

```
Between 1 And 45
```

Gaining automatic Access

A reader has written in for help: "*I'm struggling with Access 97 for NT, trying to build a database for other people to use. I have seen other databases where a form opens up automatically, which is what I want but how do I do it?*

"*Additionally, since many of my users are not used to databases, is there any way of limiting the options which are available to them when Access is running?*"

"*Finally, is there a way of getting Access to start as soon as NT fires up? Can I set up the machine so that whenever the user switches on the PC, it goes into NT, lets them log in, and then goes into Access?*"

The simple answer is yes. Let's start from the outside and work inwards. To fire up a copy of Access, you can add it to NT's StartUp menu. Click on Start and choose Settings, TaskBar and the Start Menu Programs tab. Click Add and then browse in the normal way to find Access. Then click on Next, select the StartUp folder and finish. This seems *more* complex than it used to be in Windows 3.1 — weren't operating systems supposed to be getting easier to use? (Fig 5)

Access 97 has a new startup form which allows you to control many options when it kicks into life. You can get to this by right-clicking the border of the Database window and choosing StartUp, or go via the menu with Tools, StartUp. Fig 6 shows the sort of options that are available.

The subject of passing captions from buttons has exercised the brains of several readers. For instance: "*I was intrigued by the buttons and captions capture question. Your method in the February issue appears to be the way to go and can be speeded up using [Listing 4] (using Access 7) which calls [Listing 5].*" **Kelvin**

Malcolm Fraser, too, sent in a solution and it's on our cover-mounted CD, this month, in a file called CAPTIONS.MDB. Grateful thanks are also due to Steve Foster, Richard Todd, Malcolm Bacchus, Wilf Davies, Alan Berry and others who provided elegant solutions.

Beyond the pale™

I discovered the following rumour on the web site of Bloor Research, at www.bloor.co.uk. It appears that the word Metadata™ has been trademarked™ to a pharmaceutical™ company in the US. At first sight this sounds (if you'll forgive the mixed metaphor) unbelievable, but I suppose those people who issue trademarks in the US may not be as familiar with planet database as those of us who live there. To make matters worse, the company that was awarded the trademark™ has started sending "Cease and Desist" letters to anybody it finds using, er, well, that word™. I'm currently trying to trademark the word "pharmaceutical™ " in the hopes of wreaking some small revenge™.

PCW Contact

Mark Whitehorn welcomes readers' correspondence and ideas for the Databases column, at database@pcw.co.uk.

Serves you right

Well, he promised, and here it is. As a new, regular section of his column, Mark Whitehorn introduces the subject of client-server computing explaining how, why and wherefore.

As promised last month, a section of this column will now be devoted to the topic of client-server computing.

There are several definitions used in client-server databases. For our purposes I will use the definitions shown below unless stated otherwise. What we can discuss over the next few months is:

1.Server Suitable hardware, OS and RDBMS.

2. Client Suitable hardware and software.

3. Component positioning Where you should place the business rules and the data (yes, I know I said it sits on the server, but there are some exceptions!).

We can also look at how an existing single-user system can be upgraded to client-server: that little lot should keep us busy for a few months.

Background

Most companies upsize to a client server because they need to change a single-user database into a multi-user. Although this can be done by moving the data onto a file server (see my previous columns), such solutions are usually limited in both power and the number of users. For many companies, the only really effective way to provide multi-user access to a database is by moving to client-server.

In many cases, the change to client server implies another subtle change: the database changes from a useful but non-essential part of the business to a mission-critical system. I am not suggesting that all multi-user databases are mission critical, simply that in my experience many become so. The following is a useful conversation to have with your boss before you start.

"Suppose that we go ahead and build this client-server database, and suppose that it is as successful as we all hope. Now imagine: once it has been in place for six months or so, it begins to fail sporadically. How dangerous will that be to our business?"

If the answer is: *"It will be annoying, but we can live with it because (insert appropriate answer here)…"*, you don't need to read the rest of this section. If the answer ranges from: *"Well, that would be difficult to quantify…"* to *"Such failures, if prolonged, would seriously damage the company"*, you need to think very carefully about hardware for the server and be prepared to spend some serious money.

Incidentally, you need to keep a lookout for coded, political answers like: *"We are sure that any system you build will be reliable."* This appears not to answer the question (hence the political reference) but in fact it does. It decodes as: *"We can't afford an unreliable system, and you will be fired if we get one. But, of course, we don't want to spend any extra money."* If you receive such an answer, as far as you and your career are concerned, this is now a fully mission-critical system.

If you think your client-server database is or will become mission critical, bear the following in mind. Most people are used to the fact that a typical PC costs around £1,000. What you have to do is make them realise that while that's fine for a word-processing machine, it's totally inappropriate for a mission-critical database server. For a start, database servers, by their very nature, work harder and need to be of a higher specification than a normal PC. For another thing, if a word-processing PC fails, those important letters can be typed up on another PC. If your database server fails, what are you going to run the company accounts on?

The nitty gritty

So, as you will have guessed, we have reached the part of the column where I try to get you to spend a small fortune on your server hardware. Please understand that as I try to part you from your hard-earned cash, I don't have shares in any of the hardware vendors who may be mentioned here. Rather, I just want to make sure you keep your job.

Client server definitions

■ **Standalone database** — Runs on a single machine. That machine is typically a PC running Windows and an RDBMS (such as Microsoft's Access). The data and the data-processing engine all reside on this one machine, so multi-user access to the data is not possible.

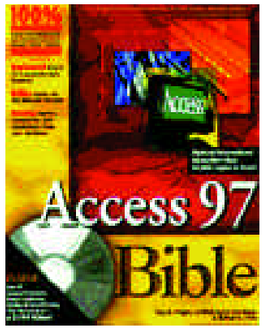
■ **Client server** — The data and the data-processing engine are moved across the network and run on a dedicated machine called a database server. Since multiple-client PCs can access this server, the system becomes multi-user. The clients will still typically run Windows and some sort of interface to the database.

■ **Client** — A PC running Windows (of whatever flavour).

■ **Server** — A computer (not necessarily Intel-based, but probably so) which runs a dedicated RDBMS back-end. This will not be Microsoft Access, since it cannot run as a database server, but instead will be something like Microsoft's SQL Server, Oracle's Oracle, or IBM's DB2. The server will also run on a server operating system such as Windows NT, OS/2 or UNIX.

p270 ➢

Book review: Access 97 Bible



Cary Prague has established himself as one of the more authoritative Americans to write about Access. His earlier books have been excellent and this one (over a thousand pages long), written in conjunction with Michael Irwin, is no exception.

It tells you how to use Access' GUI and macro language but stops short of programming in VBA (Visual Basic for Applications). Subjects are covered in detail and with accuracy. It is only when the authors stray into areas which are more to do with the relational model than Access that the content becomes a bit flaky. As an example, they attempt to distinguish between a one-to-many join and a many-to-one join. Since these are, as the authors acknowledge, essentially the same animal viewed from a different direction, it seems like needless obfuscation. It isn't helped by their statement during this rather bizarre interlude, that a many-to-one relationship is (in theory) a one-to-one relationship. Surely a misprint?

However, this minor carping on my part should not, under any circumstances, prevent you from buying this book for the treasure-chest of Access gems with which it is stuffed. For novice Access users, it will be an invaluable road-map. For experienced users, it is a wonderful source of tricks and tips. As is so often the case with books of this type, one single example or tip can save you, say, a couple of hours' work. This is a "must-have".

■ ***Access 97 Bible*** **by Cary Prague & Michael Irwin. £42.99 (IDG Books, ISBN 0-7645-3035-6) from Computer Manuals 0121 706 6000**

Also, bear in mind that, as before in this column, I will name names and quote figures, but you must only regard them as approximates. Do not base your entire business strategy on the figures I quote, because I don't know what your business requirements are. It often takes a couple of days' consultancy work to provide accurate figures for a given company. But as I hate reading evasive articles, I'll give ballpark figures which I think are reasonable for the average small-to-medium-sized enterprise (SME).

Buying a server

Buy it from a reputable company. I know they charge more but you usually get what you pay for, not only in terms of reliability but also in terms of compatibility (important with the kind of server OS you will be running) and support.

Good names here are Compaq, IBM, HP, Olivetti, Apricot, NetFrame etc. Note the "etc". Just because I haven't named a supplier, doesn't mean it produces poor products. On the other hand, don't assume just because a supplier can make a reasonable PC, it can also make a good server. Buy from a manufacturer with a good track record in making servers.

Questions and answers

I have received a few readers' letters during the past couple of months, so let's deal with these.

Q. *"Regarding the problem posed by Gareth Wade in your April column, which was about the problem of displaying times greater than 24 hours in Access. It does not have the [h]:nn:ss format that Excel uses for displaying hours greater than 24. As far as I am aware there is no format, as such, that will solve this problem. Access has other uses for square brackets.*

"I experienced a similar problem when totalling the times in a relay race that took place over two days. I realised the total time for a team could be greater than 24 hours and would then revert back to 00:00. I fixed it in a hurry by displaying just the minutes

Fig 1 There are several ways in which times can be manipulated

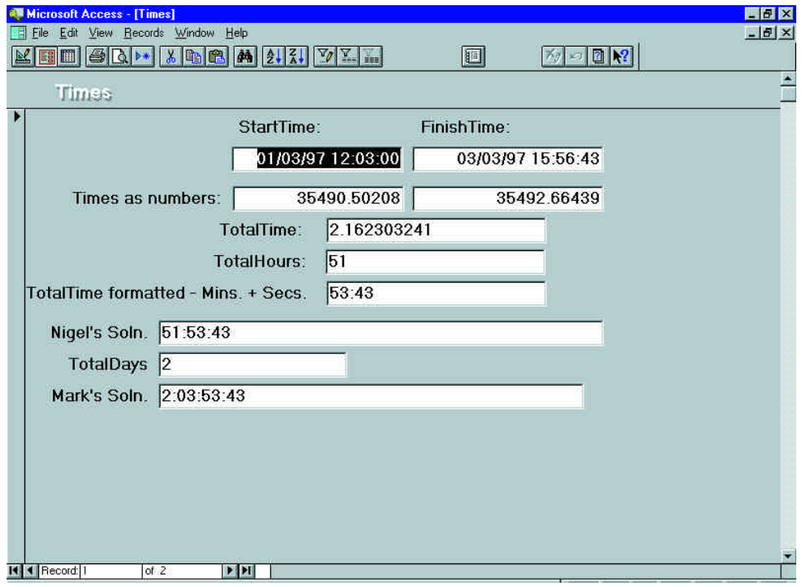
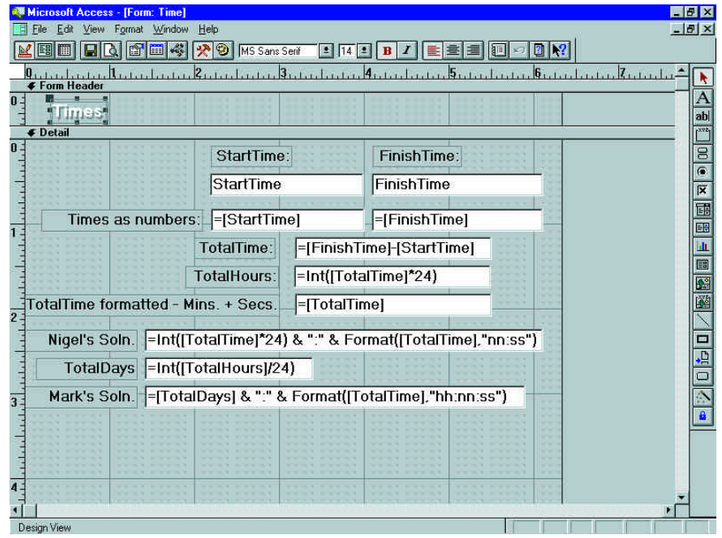


Fig 2 A design view of the same form *[see also Fig 1]*



*and seconds with nn:ss format and dealing with the hours separately. As I am sure you are aware, dates and times are stored as double precision numbers that represent the number of days after 30th Dec, 1899, so Int([TotalTime]*24) displays just the number of hours. The two parts can easily be combined into a single text box by making its control source*

```
=Int([TotalTime]*24) & ":" & Format([TotalTime], "nn:ss")
```

"While not actually being a formatting solution, this nevertheless seems to address the problem."

Nigel Collins

A. Nigel's solution is very neat. I have included a form which shows his solution, as well as some of the intermediate steps (Figs 1 & 2). The first two text boxes just show data from two fields in a record. These are the start date/time and the finish date/time of some mythical event. These text boxes are formatted as General Dates. The next two show the date/times formatted as numbers with fixed numbers of decimal places. As Nigel points out, the dates/times are really double-precision numbers that represent the number of days after 30th December 1899.

In Fig 2, you see that the syntax of the control source for these text boxes is of the type

`"=[StartTime]"` – *not* `"StartTime"`

Unless you use this syntax, Access will not allow you to choose a format like Fixed for the text box (although you can still type that format in by hand).

The third row on the form shows one text box which is calculating the difference between these two numerical values, while the fourth is calculating the total number of whole hours between the two dates.

The fifth shows the difference formatted as minutes and seconds. Formatting in this way effectively tells Access to ignore the hours and show only the minutes and seconds. (Note that this uses "nn:ss" and not "mm:ss" as you might expect.)

Nigel's solution is shown next and elegantly combines these two methods of showing the time difference. This led me to realise that we can also calculate the total number of days relatively simply and hence display the information in days, hours, minutes and seconds if so desired, as shown in the final two text boxes in Fig 2. This is clearly not *better* than Nigel's solution, it's just different.

Q. *"In your March Databases column, you mention the subject of storing hierarchical data in an SQL database.*

"The scheme you outline, which essentially uses a pointer to the next higher item in the hierarchy, is conceptually simple and very easy to understand. Sadly, as you noted, it is a pain to program with. You may be interested to know of a couple of articles written by Joe Celko. They were published in the weekly magazine Computing *(19th January 1995 and 26th January 1995 editions).*

"The concept he describes uses nested sets to represent hierarchical levels. Celko gives an example, using a simple organisation chart to demonstrate the tree structure. He also outlines a set of queries which allows you to do important things like find all the leaf nodes (those which have no further dependents) and identify the hierarchical chain of command from any individual, from the lowest to the top level.

"Please do follow this up and publish something in your column. Although relational databases are very good and very useful, there is nevertheless still a lot of data in the world which is hierarchical."

Richard Howells

A. Joe Celko is an American writer whose work on SQL I know and regard highly. Those interested in more efficient ways of storing hierarchical information would benefit from obtaining and reading these articles. Or send me some email, and if enough people are interested in the subject, I'll follow it up for you.

Q. *"I run a photographic model agency and we keep details of our models' characteristics, such as hair and eye colour, on our Access database. Sometimes we receive requests for "all girls with blonde hair". This is easy, as I can prepare a query for "blonde". However, I cannot seem to be able to request "all" hair colours. There appears to be no way of presenting a wild-card search as one of the input fields on the query through "[Input hair colour:]".*

"Can you think of any way of handling this criterion without resorting to non-Access code such as Basic?"

Mike Illes

A. I presume you're using parameter queries, in which case this syntax may be useful:

```
Like [Input hair colour:] & "*"
```

If you enter Blonde, it will find all Blondes. If you enter nothing, it will find all records. It also does a "fuzzy" search, so that you can mis-spell Blonde and still find the appropriate records.

PCW Contact

Mark Whitehorn welcomes readers' correspondence and ideas for the Databases column, at database@pcw.vnu.co.uk.

In the round

The subject of rounding in Access gets a discreet revival, and your contributions are sought for the most unusual RDBMS application. Mark Whitehorn briefs you on what's required.

This column has harboured several discussions about rounding functions in Access. I thought all had gone quiet until two intriguing emails arrived, one from Roger Moran and the other from Ray Hall. While some of us (myself included) find this topic fascinating, I am loath to devote much more space to it since it may be of limited interest to some people. So, I have included their emails in full as memo fields in the DBCMAY97.MDB file on the CD-ROM. This ensures that the information is available to those people who wish to look at it, but doesn't soak up bandwidth for those who aren't. See the form called "Rounding" if you are interested, and thanks to Roger and Ray for their contributions.

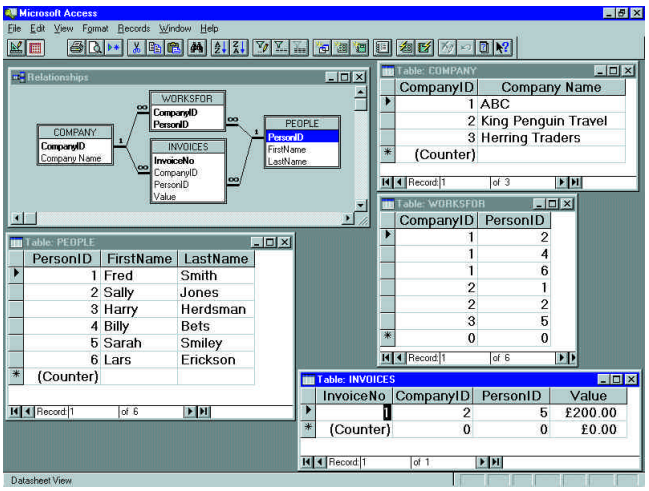


Fig 1 Taken from DBCMAY97.MDB and showing the tables from Greg Barstow's question

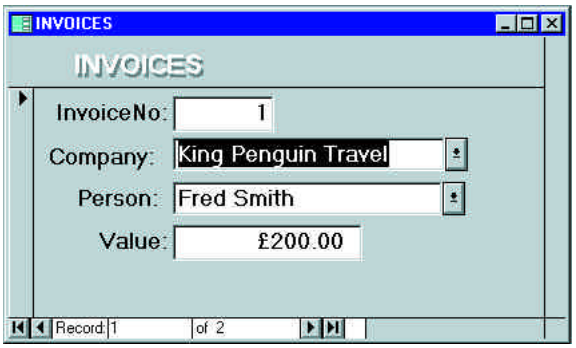


Fig 2 The two combo boxes on the form from DBCMAY97.MDB

Competition time!

From Andrew Leaman: *"I am interested in databases but find it difficult to think of useful applications. Could you possibly supply a list of typical applications, starting at the simplest and working up to the more complex?"*

Starting with the most basic database is easy: an address list, maybe a list for sending Christmas cards. As to the more complex uses, these are almost without number and range. Banking systems, air traffic control systems, process control systems in factories — all have at their heart some form of database. In fact, it is possible to argue that almost all computer applications are essentially databases; some of them just have rather odd front-ends.

Take a word processor, for example. It stores and manipulates data. You can query the database (with the search facility), generate different forms (normal view, outline view, etc.) and generate reports (print-out). You see? A document is really a database and a word processor is really a DataBase Management System.

I wouldn't want to take this argument too far, but what about a competition? We'll offer one of our much sought after book/record tokens for the most unusual example of an application developed with a recognised RDBMS. It doesn't have to be developed by you or your company, but if it happens to be so, that's fine. Just to start the ball rolling, I've heard of a really odd application, developed in Australia, which made the international news recently... But I'll leave it open for a reader to suggest that one since I'm sadly excluded from the list of potential prize-winners.

Sets and subsets

Greg Barstow writes: *"I do freelance for a range of companies. Each company has a number of people who can commission work from me for the company concerned. I need to generate invoices for each piece of work, and I want a form in Access which lets me pick the company from a combo*

p286 ➢

box (which I can do easily). Then I want the next combo box (which allows me to pick the person to whom the invoice should be sent) to show me only the people who work in that company."

This is a good generic question. Essentially it asks: "Given a long list of options which can be unequivocally sub-setted by a choice in another list, how do I show this elegantly on a form?" There are many applications. If you have different sales people who work on different product lines, or different aircraft which are serviced by different engineers, this is an area which may be of interest to you.

For one possible solution, see DBCMAY97.MDB. Fig 1 *(p285)* shows the tables involved and a small quantity of sample data. It also shows a tempting but incorrect set of relationships between the tables. Those who enjoy conundrums can work out why this particular set of relationships works but is non-optimal. The answer is on page 288 (the relationships in the MDB file on the CD-ROM are correct).

The form (Fig 2) has two combo boxes. The upper one is straightforward. It looks up values in the table COMPANY:

```
Select [CompanyID],[Company Name]
From [COMPANY];
```

and writes the value from COMPANY.CompanyID into INVOICES.CompanyID.

The lower combo box is more devious. It pulls data from a query called TheRightPeople rather than directly from PEOPLE. The purpose of this query is to find the people who work for the company which has been selected in the upper combo box.

The SQL for this query (Listing 1) is, like a great deal of SQL, impenetrable at first glance. Translated into English (more or less) it says:

- Find the value which is in the combo box above.
- Use that value to find the correct record in the COMPANY table.
- Then use that to find the people who work for the company. This has to be done via the table called WORKSFOR, since that is the table which stores the information about who works for which company.
- Finally, collect the relevant person's ID number and assemble their name neatly, attaching the first and last name together so that it looks tidy in the combo box.

If we run through that again, we can add in the relevant bits of the SQL statement:

Find the value which is in the combo box above.

```
[forms]![invoices]![companyID]
```

Use that value to find the correct record in the COMPANY table.

```
WHERE ((COMPANY.CompanyID=
[forms]![invoices]![companyID]))
```

Then use that to find the people who work for the company; this has to be done via the table called WORKSFOR, since that is the table which stores the information about who works for which company.

```
FROM PEOPLE
INNER JOIN (COMPANY INNER JOIN
WORKSFOR
ON COMPANY.CompanyID =
WORKSFOR.CompanyID)
ON PEOPLE.PersonID =
WORKSFOR.PersonID
```

Finally, collect the relevant person's ID number and assemble their name neatly, attaching the first and last name together so that it looks tidy in the combo box.

```
SELECT DISTINCTROW
COMPANY.CompanyID,
[FirstName]+" "+[Lastname] AS Name,
PEOPLE.PersonID
FROM PEOPLE
```

The nett result is that, when the second combo box is opened, the only names that appear belong to people who work for the company you have just chosen in the upper combo box. At least it *would*, except that it doesn't work automatically all the time because Access often buffers information so it doesn't automatically re-query the source for a control. The lower combo box has the query called TheRightPeople as its source. If this query has already returned an answer table, then simply selecting a different company in the upper combo box doesn't cause TheRightPeople to be re-run; hence the list of people may not be up to date when it appears in the lower combo box.

The answer is to force a re-query every time you use the lower combo box. This can be done in code or with a macro, and since I usually demonstrate everything in code, I thought for the sake of variety I'd use a macro (Fig 3).

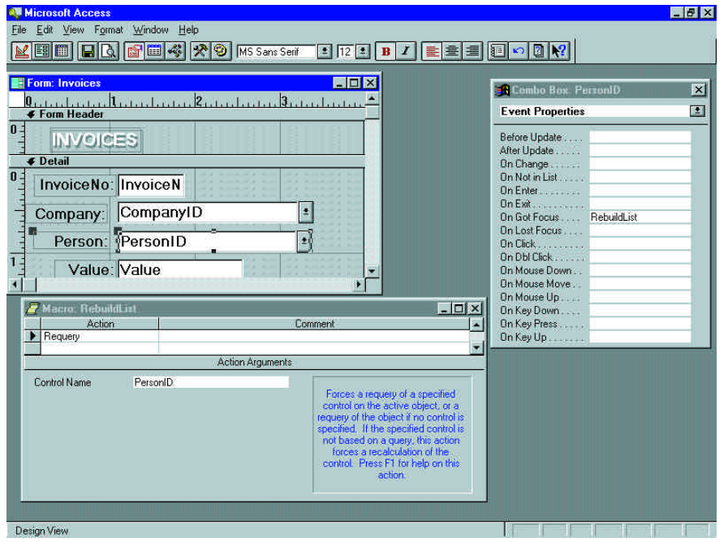


Fig 3 The extensive, and highly complex, macro used to force a re-query of the lower combo box on the form

David Ruffel emailed in a question which seems to have general application. He maintains a list of consultants who are experts in various areas — mathematics, computing, etc. There is a many to many relationship between the consultants and their subject areas, hence three tables are needed to model this relationship, while a fourth table contains information about times when the consultants are absent (Fig 4). Finding those consultants who can provide information about a given subject, say, Chemistry, presents no problem (Listing 2, p288).

However, David also maintains a table of the dates during which particular consultants are unavailable. What he needed was a query which found not only the consultants who were experts in a particular area, but also those who were available on a particular date.

To my twisted mind, the easiest way to solve this one is to use one query to find all of the Chemists who are unable to work. This can be accomplished with Listing 3 which looks pretty horrible if you aren't used to SQL, but is much more understandable in Access' GUI (Fig 4a). This produces an answer table which looks like Fig 5.

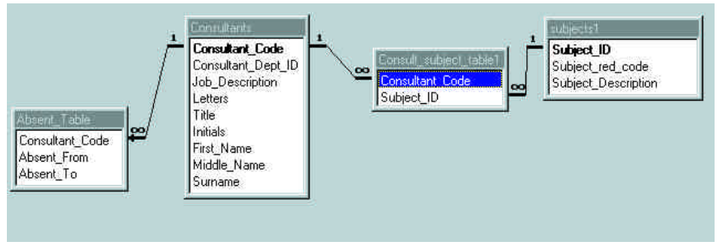


Fig 4 The tables used in David Ruffel's database

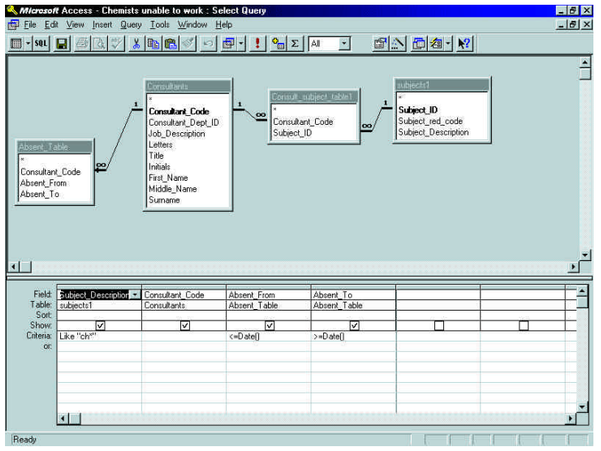


Fig 4a The GUI version of a rather impenetrable SQL statement

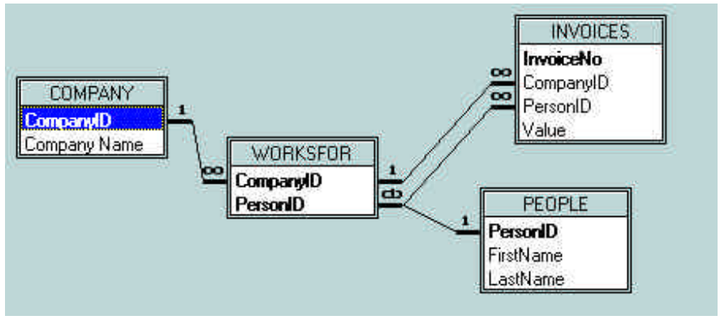


Fig 7 Showing a better set of relationships to use in DBCMAY97.MDB

The ConsultantCode identifies the Chemist who can't work on the given date (in this case I am using the Date() function to return today's date). Then, a simpler query can use the information in this answer table to identify all of those who can work (Listing 4).

This solution may not be optimal and I haven't tested it extensively, but you might want to use something akin to it if you have a similar problem.

The sample file is on the CD-ROM as an Access 7 file called QP4.MDB.

The format of this file (Access 7) brings me to another email, from Dave Milor.

Versioning

"Is there any specific reason why you write your articles with reference to Access version 2 but are using the Windows 95 interface? I have heard that there are problems with version 7 regarding speed and possible bugs."

I use Access 2.0 whenever possible simply because Access maintains compatibility in only one direction. If I provide a solution in Access 2.0, anyone using that version or later can read it. However, if I supply an MDB file in the most recent version, Access 97, only those people with that version can use it. All of the machines that I now use are running either Windows 95 or NT 4, which is why the screenshots of Access 2 appear as they do. When run under 95 or NT, Access 2 "acquires" the look and feel of these particular systems.

With regard to bugs, Access 7 certainly has them; but then, so does every bit of software I have ever seen (including my own). I haven't come across any which would make the product unusable.

The speed issue is more complex. Given any RDBMS, speed considerations can be split into two areas. First, there is how fast the interface runs. This is non-trivial, since a slow interface makes development work painful and will upset end-users. Second, there is data-processing speed: essentially the speed with which queries run. This is very different, and also clearly non-trivial. This second measure of speed is the one which people like myself love to benchmark, but we shouldn't ignore the interface speed, even if it is more difficult to quantify.

So, what about speed in Access 7? The interface speed is worse than Access 2.0, but the data processing speed is a bit better with certain queries. Access 97 (the

p288 ➢

Listing 1 SQL for TheRightPeople query

```
SELECT DISTINCTROW COMPANY.CompanyID,
[FirstName]+" "+[Lastname] AS Name, PEOPLE.PersonID
FROM PEOPLE
INNER JOIN (COMPANY INNER JOIN WORKSFOR
ON COMPANY.CompanyID = WORKSFOR.CompanyID)
ON PEOPLE.PersonID = WORKSFOR.PersonID
WHERE ((COMPANY.CompanyID=[forms]![invoices]![companyID]));
```

Listing 2 Finding Chemistry consultants

```
SELECT DISTINCTROW subjects1.Subject_Description, Consultants.Title, Consultants.Surname
FROM subjects1
INNER JOIN (Consultants INNER JOIN Consult_subject_table1
ON Consultants.Consultant_Code = Consult_subject_table1.Consultant_Code)
ON subjects1.Subject_ID = Consult_subject_table1.Subject_ID
WHERE (((subjects1.Subject_Description)="Chemistry"));
```

Listing 3 Finding the Chemists who are unable to work

```
SELECT subjects1.Subject_Description, Consultants.Consultant_Code, Absent_Table.Absent_From,
Absent_Table.Absent_To
FROM subjects1
INNER JOIN ((Consultants LEFT JOIN Absent_Table
ON Consultants.Consultant_Code = Absent_Table.Consultant_Code)
INNER JOIN Consult_subject_table1 ON Consultants.Consultant_Code = Consult_subject_table1.Consultant_Code)
ON subjects1.Subject_ID = Consult_subject_table1.Subject_ID
WHERE (((subjects1.Subject_Description) Like "ch*")
AND ((Absent_Table.Absent_From)<=Date())
AND ((Absent_Table.Absent_To)>=Date()));
```

Listing 4 Identifying the Chemists who can work

```
SELECT DISTINCTROW [Able Chemists].Consultant_Code, Consultants.First_Name, Consultants.Surname
FROM Consultants
INNER JOIN [Able Chemists]
ON Consultants.Consultant_Code = [Able Chemists].Consultant_Code
WHERE ((([Able Chemists].Consultant_Code) Not In (Select Consultant_Code from [Chemists unable to work])));
```

Fig 5 Answer table from Listing 3

Subject Area Description	ConsultantCode	Away From	Until
chemistry	5	Thursday, January 02, 1997	Thursday, June 19, 1997

next version on) definitely requires a better machine to run the interface (in other words, it is even slower). However, the data-processing is markedly faster, even given machines of the same spec.

Fig 6 Entry into the INVOICES table

InvoiceNo	CompanyID	PersonID	Value
1	1	3	£200.00
2	1	4	£0.00

A client-server future

In the February issue I asked if the column was too biased towards Access and, more generally, what did people want me to cover in future issues. The area which was overwhelmingly popular as a new topic was practical information regarding client-server databases.

This email from Tom Cliff was typical:

"I've been building databases for a couple of years and all of them have been on standalone PCs. Some of these have grown and now need to be migrated to a client-server system, because the volume of data has increased or they need to become multi-user. What I need is a good overview/ description of what is involved. How much work is it? How do I actually do it?"

So, we'll start next month having a look at the hardware you need, then move on to actually installing a back-end RDBMS, setting up a database on it, connecting to that database from clients, and so on.

Answer to conundrum

The relationships shown in Fig 1 ensure that the values inserted in INVOICES.PersonID are drawn from the available list of people (which is kept in table PEOPLE). They also ensure that the values inserted into INVOICES.CompanyID are drawn from the list of available companies. What these relationships *don't* forbid is an entry into the INVOICES table like Fig 6. This is bad, because person 3 doesn't work for company 1.

A much better set of relationships to use are those shown in Fig 7 *(p287)*. These ensure that the INVOICE table can only ever contain "meaningful" combinations of CompanyID and PersonID.

PCW Contact

Mark Whitehorn welcomes readers' correspondence and ideas for the Databases column, at database@pcw.vnu.co.uk

All buttoned up

Mark Whitehorn presents another slant on last month's buttons and captions query, which is far from sewn up. Plus, the complexities of data entry in Access, and CUSTOMER care.

In the February issue I published a letter from Glen Rowe. He wanted multiple buttons on a form, all with different captions. Whenever a button was pressed, a query would run which returned values based upon the caption on the button. For example, if you press the button labelled "Penguins" you see the records which relate to Penguins: pressing the "Fish" button yields information about Fish. I produced a solution by building a query which snatched the caption from the button which had just been pressed.

I also wrote the following: "The obvious solution at first is to try to pass the button's caption to the query as a parameter. As far as I know (and I stand to be corrected) this can't be done."

James Talbut replied: "Passing a parameter to a parameter query in Access can be done, but it's not very pleasant and means the you need to use DAO (Data Access Objects), which can be good or bad depending on the circumstances."

The solution that James then provides does, as he suggests, pass a parameter to a parameter query. What it doesn't do is pass the *caption* of the button as a parameter (which is still, as far as I know, impossible). In turn, this means his solution is somewhat more awkward than the one shown in the February issue because every button you add to the form must have its caption *and* its OnClick property altered. With the February solution, all you had to do was clone the button and alter the caption. However, James's solution to the initial problem is still well worth studying since it illustrates a very different way of solving the conundrum.

He continues: "Before I include all the bumf to show you how it's done, I'll just mention the implications.

"To actually access the resultant data you need to use DAO to step through a recordset. If you have a large amount of data this can be inefficient (OK, it's always inefficient, but it's more noticeable with a large dataset), particularly if you are intending to perform some secondary operation/selection on the data. However, the query itself is still run by the JET engine so the main data manipulation routines are still run as quickly as Access can do them.

"To control the parameters you need to use the Parameters collection of the QueryDef object for the query you are interested in. The most simple example I could come up with looks like this:

Table: "People"

ID	Name	Value
1	Fred	1
2	Jack	4
3	Fred	9
4	Harry	16
5	Anastasia	25
6	Bob	36
7	Martin	49
8	Stephen	64
9	Harry	81

(See Fig 1.)

```
Fig 1 Query, "Person":
SELECT DISTINCTROW People.ID, People.Name, People.Value
FROM People
WHERE (((People.Name)=[Person]));
```

```
Fig 2 Form Code, behind "People"
Private Function Button_Click(sName As String)
Dim qdfPeople As QueryDef
Dim rsPerson As Recordset
Dim sRowSource As String

Set qdfPeople = CurrentDb.QueryDefs("Person")

 qdfPeople.PARAMETERS("Person") = sName
 Set rsPerson = qdfPeople.
OpenRecordset

 While Not rsPerson.EOF
   If Len(sRowSource) > 0 Then sRowSource = sRowSource & ";"
   sRowSource = sRowSource & rsPerson.Fields("Name") & " - " &
rsPerson.Fields("Value")
rsPerson.MoveNext
 Wend
 List10.RowSourceType = "Value List"
 List10.RowSource = sRowSource
End Function
```

p286 ➢

Form, “People”:
Six command buttons, each with their OnClick property set to:

```
=Button_Click(“Fred”)
```

with Fred replaced by the caption on that particular button. The buttons are labelled Fred, Bob, Harry, Martin, Anastasia and Stephen. At the bottom is an empty list box called List10.
(See Fig 2, page 285.)

“And that’s it. Hope it’s useful.”

Very interesting, James. I have constructed an example file (in Access 7.0) based on this example called TALBUT.MDB which is on the CD.

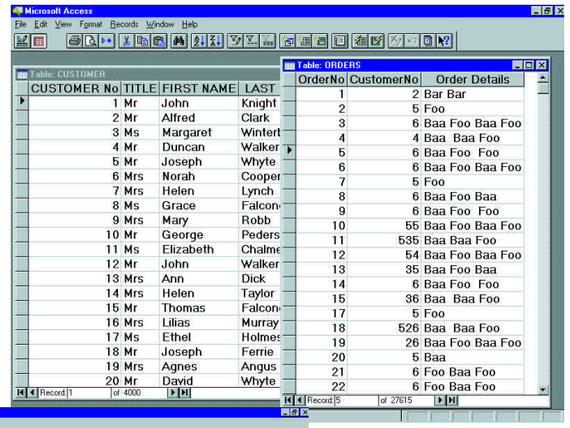
Next an email from Norway: “I have a problem that I’ve worked on for several months now, without finding any easy solution,” *writes Tore Saetre, of Bergen*. “I’ve developed and am maintaining a Microsoft Access database that keeps track of customers. The database contains two main tables: Customer and Sales. The problem is that I need to have a field in the Customers-table that shows how many orders the customer has in the Sales-table.

“My first idea was to make a query that counts orders in Sales for each customer. The query lists each customer and how many orders the customers has. I tried to move that value to the Customer-table with an update query that updates a NoOrders-field in Customer through a link to the first query. The problem is that the first query can’t be updated and the second query won’t update my Customers-table.”

I receive many questions like this one, questions which hinge upon the desire to store redundant data in tables. My initial reply to Tore was that, in general, storing derivable data is a bad idea because if the data changes (as you make more sales), the data in the CUSTOMER table goes out of date. It is normally preferable to use a query to calculate the information you need: whenever you want to see this data, you can run the query which shows you the customer details and the number of sales.

Tore was only partially convinced and replied: “But data in the sales table is only changed and added to once a month. The rest of the month the user browses through CUSTOMER’s 4,000 records and needs to see as much data on each customer as possible. Running a query for each customer is too time-consuming. I see why it can’t be done in a ordinary customer/sales database, but I still hope to find a solution to this problem.

Fig 3 (right) Small samples from both the CUSTOMER and ORDERS tables in the database “Tore”



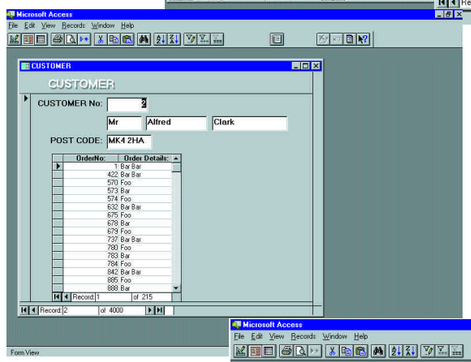
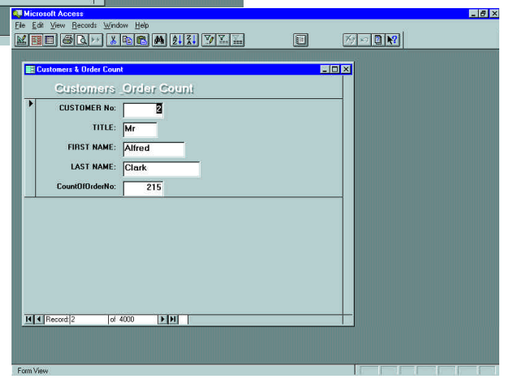


Fig 4 (left) The form called “Customers & Orders” shows customer details and order details

Fig 5 (below) The form called “Customers & Orders Count” shows one record for each customer



“For an experiment, do you have an example of how I can run a query to see customer details based on the customer currently displayed in a form? Do I need to run a code, macro or function and how do I get the query to filter everything but the record on-screen? Thanks for your help.”

Okay, we’ll solve this one both ways. There is a sample database on the cover CD (in Access 2.0) called TORE.MDB. This has two tables ,CUSTOMER and ORDERS. CUSTOMER contains simple details of 4,000 mythical people, ORDERS contains about 27,500 orders. Each customer has at least three orders to their name (or, in this case, to their CustomerNo) and some have considerably more. Fig 3 shows small samples from both tables.

The form called “Customers & Orders” (Fig 4) shows customer details and order details. The main form is based on CUSTOMER, the sub-form (which is called Sub1) is based on a query which simply performs a join on the two base tables. This form displays one record for each of the 4,000 customers.

The form called “Customers & Orders Count” (Fig 5) again shows one record for each customer. However, instead of listing the details about each order, it simply shows how many orders each customer has placed. This form is based upon the query called “Customer & Order Count”.

These two forms are variants which answer Tore’s request for an “example of how I can run a query to see customer details based on the customer currently displayed in a form”.

I ran this database on a 486/100 with 16Mb of RAM using Access 2.0. “Customers & Orders” opens almost instantaneously, and you can scroll through the records at the rate of about 400 per minute. “Customers & Orders Count” takes about 25 seconds to open, but once open you can scroll through the records at the rate of about 1,600 per minute.

To put this into simplistic terms, the first one does the calculations for each record as you ask to see it (which is adding about 0.1 seconds to the scroll time between each pair of records). The second one does all of the calculations for all of the records before showing any of them to you.

These forms are simply based on queries: no code or macros are required. As you can see, using the first form, it is possible to see customer and order details with essentially no speed hit at all for these numbers of records.

The query called “Customers & Orders Count” (the one which takes 25 seconds to run) can also be used, if required, to create the base table that Tore requested. I have included a Make-Table version of the same query in TORE.MDB. If you run this, it will create a base table called CUST which is just like CUSTOMERS except that it includes a field which shows how many orders each customer has placed.

In Tore’s case this could be run once a month, and then CUSTOMERS replaced by CUST; in that case, the 25-second speed hit disappears. The downside is that we are now reliant upon redundant data in the database. If anyone forgets to renew the tables at the end of the month, then all of the users of the database start to work with inaccurate data. You pays your money, you takes your choice — speed or security.

Quickies

■ Here’s a quick one from Gareth Wade: “When running a simple Report, I need to add all the time lengths and give a running

Fig 6

```
Private Sub LastName_AfterUpdate()

End Sub
```

Simply add a line so that it now reads;

```
Private Sub LastName_AfterUpdate()

    Me!LastName = StrConv(Me!
LastName, 3)

End Sub
```

Fig 7

```
Me!LastName = StrConv(Me!LastName, 3)
```

Fig 8

```
Private Sub LastName_AfterUpdate()
Dim Length As Integer

Me!LastName = StrConv(Me!LastName, 3)
If Left(Me!LastName, 3) = “Mac” Then
   Length = Len(Me!LastName)Length - 3)
   Me!LastName = StrConv(Me!LastName, 3)
   Me!LastName = “Mac”  Me!LastName
End If

If Left(Me!LastName, 2) = “Mc” Then
   Length = Len(Me!LastName)
   Me!LastName = Right(Me!LastName, Length - 2)
   Me!LastName = StrConv(Me!LastName, 3)
   Me!LastName = “Mc” + Me!LastNameEnd If

End Sub
```

p288 ➢

total. Easy enough; but when Access gets to 23:59, it reverts back to 00:00. What format do I use to carry on past that magic 24-hour mark ? I have tried all the manuals and help files but can't find the answer."

I don't know the answer to this one. Anyone else care to solve it?

■ And this one from Malcolm Rowley: "I find I am put off by the complexity required in Access to solve simple problems, e.g. data entry character formatting.

In Alpha I simply have to specify the word/character format I require, e.g. Upper, Lower, Word (first character upper case, the rest of the data left as is) in the field rules, and this is applied to all data entry that takes place and can be re-applied to all existing data. This is ideal for data entry of names and addresses; the only difficulty being the McCartneys (etc.) of this world. To have any simple form of formatting that stores the data in an access table in a particular format, do I really have to resort to code? Is there a simple way of accepting data entry in Access and storing it as first character upper case, the rest left as is?

For example: macdonalds becomes Macdonalds / macIntyre becomes MacIntyre / 27 park avenue becomes 27 Park Avenue. A simple solution would be appreciated."

The answer is that you have to use code, but only one line so it may just meet your requirement for simplicity! Suppose you have a form called Capital which displays a field called LastName in a text box called LastName. Switch to design, double-click on the textbox in question, select the Event properties, click on the one called "After Update", click on the ellipsis button which appears (three dots) and choose Code Builder. Between the two lines which appear (Fig 6) "Me" means the current form, "LastName" is the name of the textbox, and "StrConv" is a function (only available in Access 7.0 and above) which converts strings. The "three" tells the string conversion function to do the type of conversion that you asked for (capitalising the first letter of each word).

So, Fig 7 means "make the contents of the textbox called LastName equal to the same as it is now, but with all of the first letters of the words capitalised". This will convert: penguin penguinsson to Penguin Penguinsson / 23 the larches to 23 The Larches / mcdonald to Mcdonald and so on.

■ On a slightly different tack, I haven't used Alpha for well over a year, but I agree that

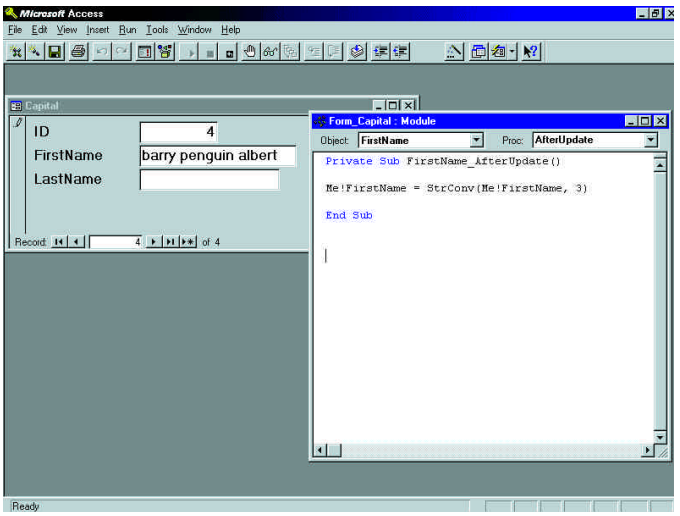
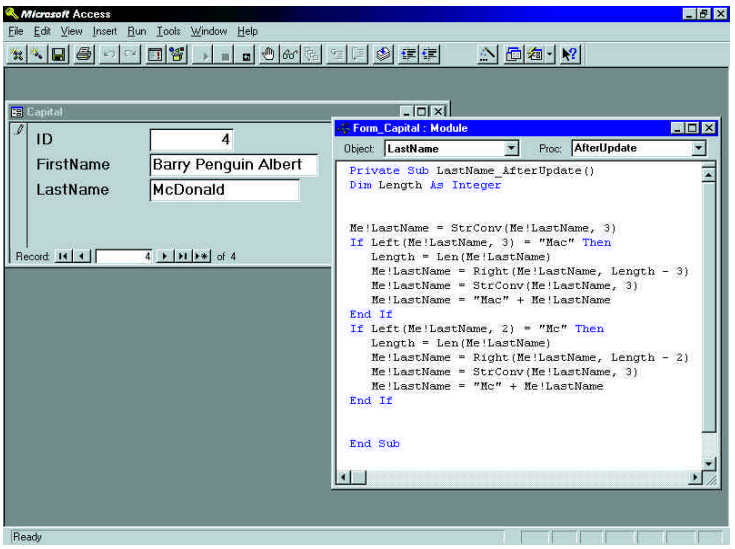


Fig 9 (left) This single line of code will capitalise all distinct words placed in the FirstName field *(see Fig 10)*

Fig 10 (below) This more complex code capitalises two of the more complex surname types in the LastName field



processes like this one are easier (and certainly more intuitive) in Alpha than in Access. In fact, to quote from a review I wrote at the time of its release: "…of these, the Field Rules are the most impressive, if only because Alpha copes with them better than any other RDBMS I have seen."

There is clearly a trade-off in design terms, which these two products exemplify. In both products, the designers identified a host of commonly needed functions and made them easily available from the interface. This approach has pros and cons. The good news is that commonly performed processes (like designing tables and building queries) are easy: the bad news is that if you happen to want a function that the designers didn't provide, you suddenly have to go to a more complex area, such as coding.

One difference between the two products is where the line was drawn. In this case, the Alpha designers decided to include capitalisation in the "easy" set and the Access designers didn't.

I have always found that, whatever product I use, I eventually reach a stage where I have to use code simply because it is impossible for the designers to put everything into the "easy" set. On a happier note, once you do start to use code, your horizons expand considerably. You can, for example, begin to deal with unusually capitalised names like those you mention. If you expand the code to that shown in Figs 9 and 10, this will change macdonalds to MacDonalds, and mctavish to McTavish. (Also see the Access 7.0 sample database called ROWLEY.MDB.)

I am not suggesting that this is perfect code. As usual in the sample code I give, there is no error trapping. In addition, there are some people who prefer the letter after Mac or Mc to be left in lower case. This code is simply provided as an example of what can be done.

PCW Contact

Mark Whitehorn welcomes readers' correspondence and ideas for the Databases column at database@pcw.vnu.co.uk

Animal magic

Mark Whitehorn casts a beady eye over hierarchy in the zoo and shows a way of handling categorisation data. In addition, he passes on some improved solutions to past problems.

A reader called Andrew writes: "First of all, may I say how much I have enjoyed your column in *PCW*" *(ah, I love flattery — MW)*. "But unfortunately it has got me thinking about a problem which I have never satisfactorily resolved for myself." *(Sounds like work! — MW)*. "It involves a recursive link field in a table:

Field	Name	Type	Description
1	Code	Inc	Primary Key
2	Parent	LongInt	Parents Key
3	Category	String	Category Title

"The idea is to provide a hierarchy by using the table to emulate a tree structure which could be used for categorising animals, or departments in an organisation.

"I have been searching for an efficient way of finding all the categories that are below the current item. My best effort has been to run a query that finds all records whose parent is current code, add these to a result table and then query the answer to find all the records where there is a match on any of the codes. I keep this going until there are no records returned (when I've reached the bottom of the tree). Although this works, it does not seem very efficient. Do you have any ideas?"

What I have done is to look at ways of handling data from a table like the one Andrew describes. Fig 1 shows such a table. It describes a hierarchy which is used to name the animals in a zoo.

Each record simply represents a single link in the structure, so the data is stored in a reasonably economical manner. However, you need to look at several records to trace back up the structure.

So, for example, record 12 tells us that Harry's parent record is number three, which in turn tells us that Harry is a King. This in turn leads us to record two, and hence to record one, by which time we know that Harry is a King Penguin, which is a species of bird. The storage is economical, but a little unwieldy.

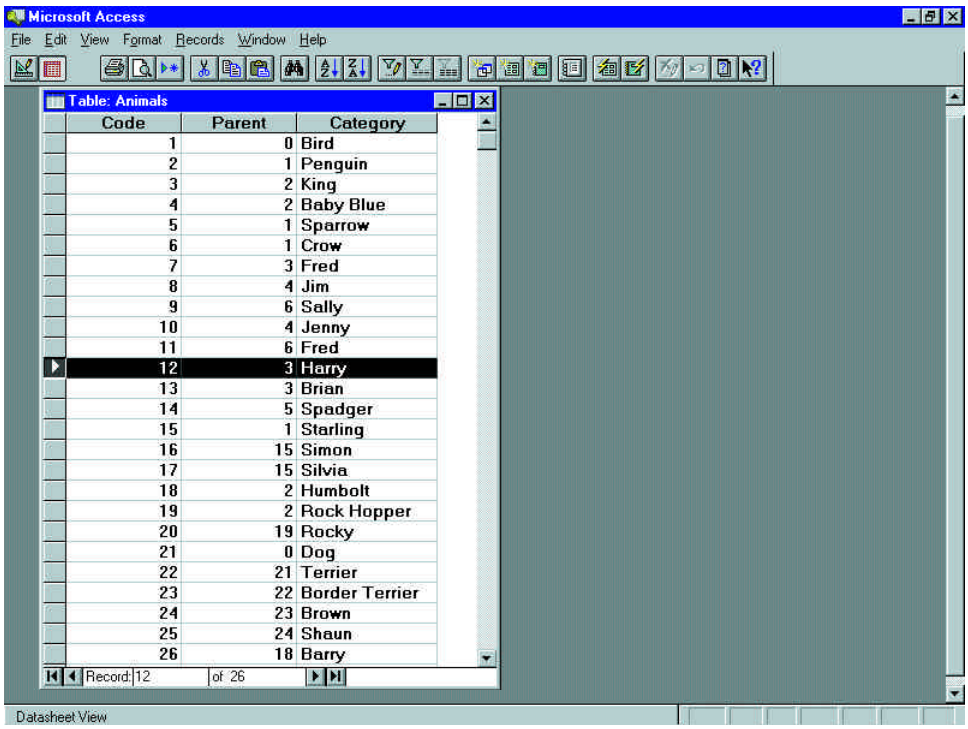
Fig 2 illustrates the information in a more legible state, and Fig 3 shows the query. Note that the answer table is not showing one record for each record in the base table. This is because it takes several records to define one "branch" of the tree. The SQL is a little tortuous, but essentially it is merely an expanded version of this:

```
SELECT DISTINCTROW Animals.Category
AS First, Animals_1.Category AS
Second
FROM Animals AS Animals_1
RIGHT JOIN Animals ON
Animals_1.Parent = Animals.Code
WHERE ((Animals.Parent=0))
ORDER BY Animals.Category,
Animals_1.Category;
```

which works for two levels of hierarchy.

This naming hierarchy is a little anarchistic in that names can appear at several levels. Suppose we have a more defined structure which must have, say, four levels. This will use the same base-table structure, but it provides a little more

Fig 1 Table of animals' names



p296 ➢

scope when it comes to defining forms. Two possibilities, one of which makes use of combo boxes, are shown in Fig 4, but neither is entirely satisfactory.

If anyone wants to try improving them, be my guest. The work so far is in the MDB file called DBCMAR97 on our CD-ROM.

Finally, it is possible to count those categories which exist under any given one, using a GROUP BY query based on the query (in this case called Project) which displays the data in the manner seen in the top right of Fig 4. So

```
SELECT DISTINCTROW Project.Group,
Count(Project.Priority) AS
CountOfPriority
FROM Project
GROUP BY Project.Group
ORDER BY Count(Project.Priority)
DESC;
```

produces the answer table for the data shown:

Group	CountOfPriority
Prototyping	4
Design	4
Repair	2
Year 2000	1
Modification	1
Consultancy	1

Rounding out

On to a little housekeeping. Over the past few months I've published several problems with solutions and asked for comments and/or improvements. This month, we'll tidy these up. Although most of the problems originate in Access, the solutions are usually applicable in most RDBMSs.

Last November I published an algorithm for rounding in Access. In last month's column I used several answers (of differing efficiency!) from readers. James Talbut came back with a modified version of his: "I'm probably too late, but anyway, I think I have fixed that rounding code:

```
Function Round(dNumber As Double,
iNumDigits As Integer) As Double
Dim dFactor As Double
Dim dTemp As Double
  dFactor = 10 ^ iNumDigits
  dTemp = dNumber * dFactor
  If 2 * dTemp = Int(2 * dTemp) And
Int(dTemp) <> dTemp Then dTemp =
dTemp + dTemp / 2 /
  Abs(dTemp) Round = (dTemp \ 1) /
dFactor
End Function
```

"The solution was simple. I just had to stare at it until my eyes ached. My fudge factor was either 1 or -1, depending on the sign of dTemp. Unfortunately, this was compounding the way in which Access alternates between rounding up or down. The solution was simply to fudge by 0.5 or -0.5. It's still hideous, for something as simple as a rounding function. But as a technical exercise I think it's kind of neat. And it's probably still quicker than converting to and from a string."

I agree. It's ridiculous that we have to write something like this just to round a number. Why doesn't Access simply include a rounding function? Microsoft, please note for future versions.

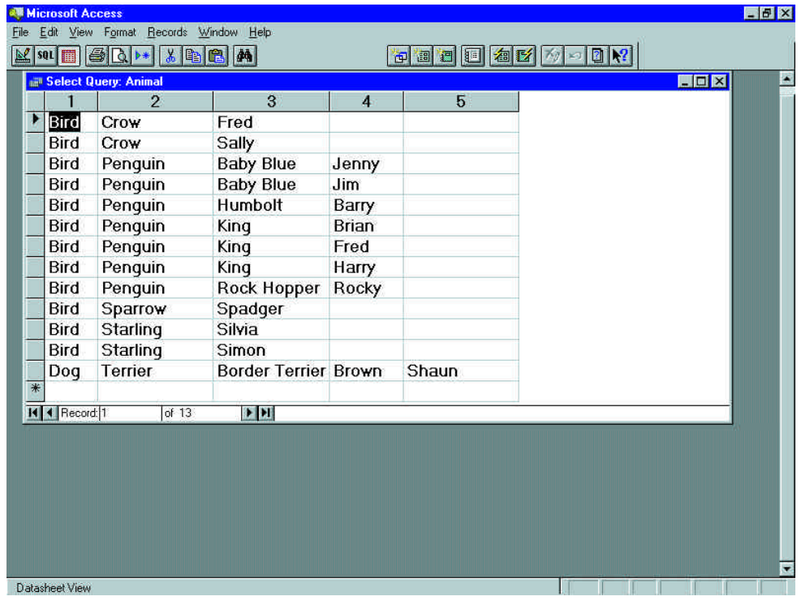


Fig 2 **The data is stored in a reasonably economical manner**

Case-sensitive joins

In last December's edition of my column, I published an email from Andrzej Glowinski who bemoaned the lack of case-sensitive joins in Access. I published a solution but he found it too slow, so I asked for other ideas as well as solutions in other RDBMSs. I was impressed with the intriguing answers, reproduced here.

On the subject of case sensitivity and the lost ninth data type, Stephen Parry writes: "There is another solution to Andrzej's problem with case sensitivity in joins, indices and sorting. It does, however, rely on a partially documented feature of access, data type 9 (aka BINARY). The BINARY field data type has been in Access since its earliest beginnings but has been omitted from key places in the package and its documentation throughout. Various system table fields use it and it appears as the data type for certain field types, in tables attached from other database products.

"It behaves much like TEXT in all respects except any comparison operations (joins, indexing, sorting) operate on ASCII code value, i.e. case sensitively. Very useful; but how do you create one? 'Not easily' is the answer to that. BINARY does not appear as an entry in the field type list in table design view. If you try to use DB_BINARY with the CreateField method in A.B. you get an error. Not very helpful. The way around this is to use a CREATE TABLE data definition query:

```
CREATE TABLE(MyField BINARY (30))
```

"This creates a table with a single field (column) called MyField of type BINARY and a length of 30. Type this into the SQL view of an empty data definition query, execute it and it will create such a table. Remember to refresh the table list in the database window to show the new table. You can then edit the table definition in table design view as per normal. You can even change the properties of the binary column(s). The data type column correctly indicates a field type of binary. You can change this, of course, but once changed you can't get it back!

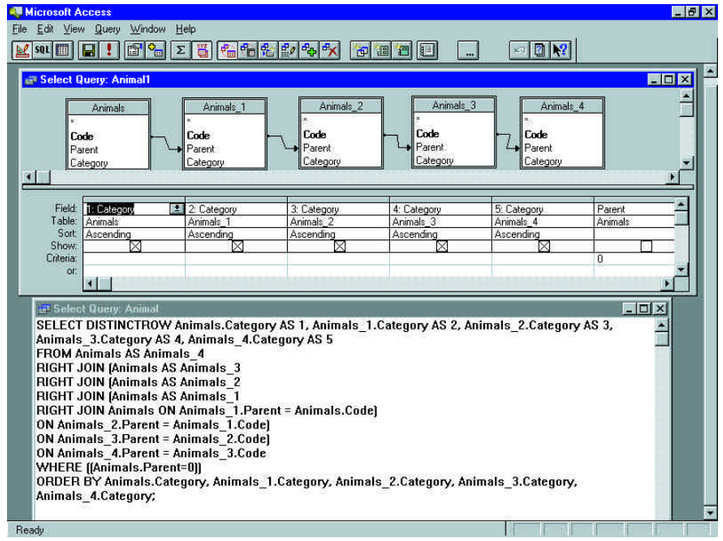


Fig 3 **This SQL looks a bit complex, but it is just an expanded version of the code listing on p295**

"To add a binary column to an existing table, I suspect an ALTER TABLE statement should work, although I have not tried it. This would allow you to add a new binary column to an existing table. You could then use code or query to copy the data from an existing column and then remove the original column. Thus, you have a long-winded means of changing a column's data type to binary.

"Of course, this all begs a fundamental question: why the (*beep!*) is something as useful as this so well hidden? I suspect that with sustained use, various 'holes' might be found in the functionality of this data type.

"Microsoft obviously intended a more complete but as yet unrealised solution to problems like this, but needed a quick solution to the Paradox and dBase connectivity problems, as well as a quick fix for some internals. For now, however, I will certainly continue to use it.

"By the way, I have noticed that in Access 7, nasty ol' Microsoft has encrypted its Wizards and other .MDA add-in files in this release. I have derived many useful programs from hacked versions of the Access 2.0 wizards, e.g. a database object directory comparison tool and a scripting tool based on the documentor, which just outputs a mammoth text file without generating a report or a temporary table. Hence, I was miffed to find that A7 has all its wizards compiled/encrypted in some way. Have you seen any way around this? I vaguely recall seeing an MSDN article on securing add-in code and possibly one giving the unsecured contents of certain wizards, but I have not been able to find either again."

I knew of the binary data type but hadn't thought of it as a solution to this problem. As to decrypting the Access 7.0 wizards, has anyone else found a way?

On the subject of case sensitivity, the following came from Neil Howie: "A bell rang at the back of my mind, so I set about creating a similar query in Paradox 7 but taking advantage of the fact that if you have maintained secondary indexes, you can set a case-sensitive option.

"Adding an auto primary index to each table and setting up secondaries on the Names fields let me set up the join in the query window and display the required result with the greatest of ease.

"For further investigation I took a long doc, converted it to text and wrote a little routine to extract the words to create two files, 12,000 records long. I modified the second to capitalise the last letter of nine words out of ten at random, then pushed these into Paradox. Because of repetitions, it produced 15,000 matches in 18 seconds, so I guess on your enquirer's 200,000++ records it would still not be a practical proposition if he had to do it too often.

"However, what is now worrying me is that executing the same query in Delphi (using Paradox's SQL) takes almost 50 seconds. What am I doing wrong?"

p298 ➢

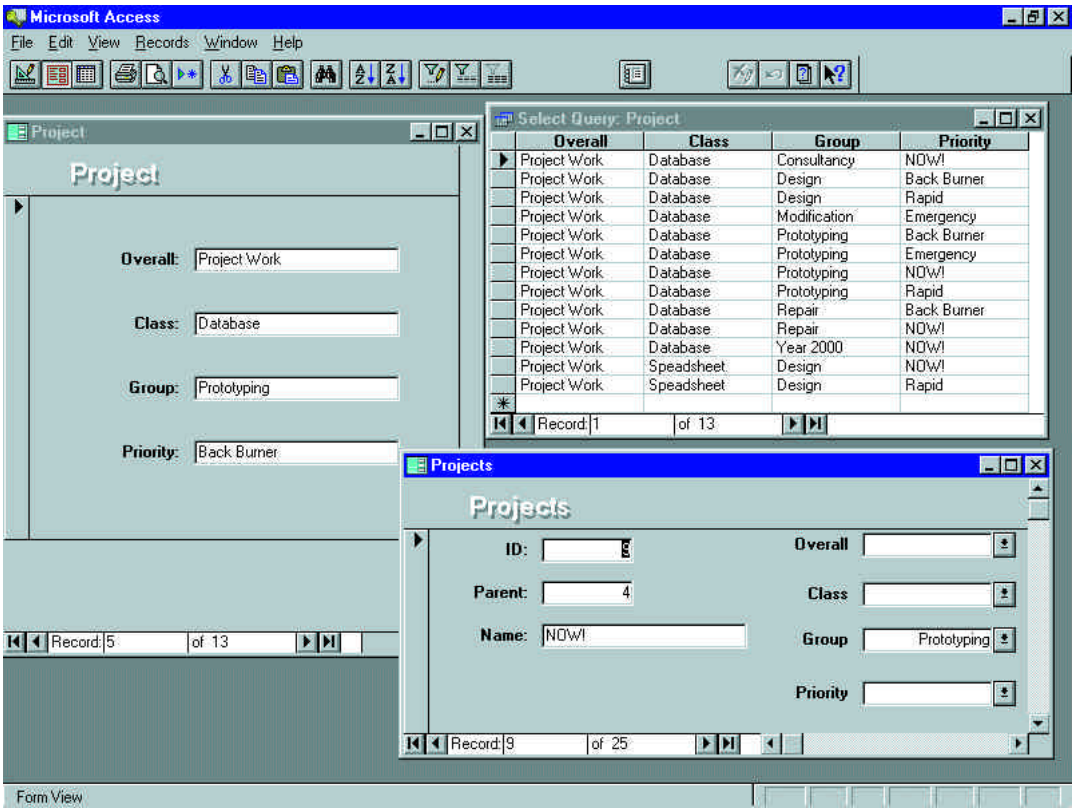


Fig 4 A defined structure with four levels provides more scope when defining forms

Making your mark

Last month I published a problem from Andy, a teacher who was using Access 2.0 to store his pupils' marks. He wanted not only to store the marks that his students achieved in their tests, but also their positions in the various class groups.

I suggested there was a conflict here between relational theory and expediency, and asked for suggestions and restraint (in the hope of avoiding a holy war). I am delighted to report that (nearly) everyone kept their heads and the majority of answers were helpful rather than religious. Paul Mapstone's answer (below) was the most complete. It is also applicable to any situation in which a rank order is required.

"My advice on this is not to store derived data (i.e. the Position column) unless you have to, for performance reasons. This is a good example. It is fairly straightforward to calculate the Position column in standard SQL using a correlated subquery as follows:

```
SELECT A.[Pupil ID], A.[Test ID],
A.Score, A.Position,
  (select Count(*) + 1
  from [Test Scores] AS B
  where B.[Test ID] = A.[Test ID]
and B.Score > A.Score ) AS Rank
FROM [Test Scores] AS A
WHERE A.Score is not null
ORDER BY Score DESC
```

"Column Rank in the above query should correctly return the required Position. This works because the Position is equal to the number of people who have a better position + 1. Alternatively, you can use the SQL'92 outer join syntax (which Access seems to partially support) as follows:

```
SELECT A.[Pupil ID], A.[Test ID],
A.Score, A.Position,
Count(B.Score) + 1 AS
Rank
FROM [Test Scores] AS A LEFT JOIN
[Test Scores] AS B
       ON A.[Test ID] = B.[Test
ID] and A.Score < B.Score
WHERE A.Score is not null
GROUP BY A.[Pupil ID], A.[Test
ID],
A.Score, A.Position
ORDER BY A.Score DESC
```

"We need the outer join, as the inner join will eliminate the top result. Either of the above queries could be used as the basis of any required reports etc, but if your teacher really wants to store the rank in the Position column (and suffer potential update anomalies), simply save one of the above queries with the name 'Rank query' and use it in the following UPDATE query:"

```
UPDATE [Test Scores]
SET Position = dlookup("Rank",
"Rank query",
      "[Pupil ID]=" & [Pupil ID] &
 " and [Test ID]="
& [Test ID])
WHERE Score is not null;
```

An excellent answer. Paul's first paragraph touches on the heart of the conflict. Storing derivable data usually has no benefits and several major disadvantages (such as causing potential update anomalies). However, storing such data can occasionally yield a major performance benefit. Of course, a real purist would never consider mere performance as a justification for breaking one of the central tenets of the relational model. Non-purists, on the other hand, wouldn't even hesitate. I sit uncomfortably on the fence, sticking to the purist line whenever possible and worrying every time expediency forces me to break what I know to be a sensible rule.

Paul's solutions are on this month's cover-mounted CD-ROM as PAUL.MDB and PAUL95.MDB. The first is an Access 2.0 version which crashes Access every time I try to run the update query. There cannot be anything fundamentally wrong with the solution because the problem does not occur in Access 95. So perhaps it is my machine?

PCW Contact

Mark Whitehorn welcomes readers' correspondence and ideas for the Databases column at database@pcw.vnu.co.uk

Making the distinction

Mark Whitehorn on the peculiarities of DISTINCTROW in Access. Plus, if you're developing a large application, see how code and components can be versatile and non-specific.

I've written this before, but it bears repeating. This column has evolved to a state where it is almost exclusively about Access. When I write about general issues (normalisation, SQL, and so on) I try to keep it as generic as possible but the fact remains that, essentially, all the questions I receive (far more than appear on the page) refer explicitly to Access.

There are probably two reasons. Either everyone who reads this column is using Access, or people who are using another RDBMS read the column and think "Huh. That's all about Access. No point in writing in about anything else."

I'm happy to accept any suggestions for the future direction of topics covered and platforms used. So if you want changes please let me know and minority interests can be represented.

Distinctly exact (or exactly DISTINCT)

In my series about SQL (*Hands On Workshop, PCW Oct'96-Jan'97*) I said that "the generic DISTINCT becomes DISTINCTROW in Microsoft's Access". By that I meant that if you build a query in Access using the GUI, then by default it will generate an SQL statement which starts:

```
SELECT DISTINCTROW….
```

In many cases this will return the same record set as an SQL statement which starts:

```
SELECT….
```

or even

```
SELECT DISTINCT….
```

The differences between SELECT and SELECT DISTINCT are explained in my workshop series on SQL, and since that was to do with generic SQL it didn't seem to be the time or place to go into the peculiarities of DISTINCTROW. But this seems like an excellent place, so we will do so! In order to demonstrate this we need a couple of tables:

CLIENTS

ClientID	Name	Location
1	Fred	Wellington
2	Sally	Wimpy
3	Jean	Halifax
4	Fred	Lancaster

ORDERS

OrderNo	ClientID
1	1
2	4
3	4
4	4
5	3
6	3
7	3
8	1
9	1

Note that we have two clients called Fred and that Sally has yet to place an order with our company.

Both

```
SELECT Name
FROM CLIENTS;
```

and

```
SELECT DISTINCTROW Name
FROM CLIENTS;
```

return four records:

Name
Fred
Sally
Jean
Fred

whereas

```
SELECT DISTINCT Name
FROM CLIENTS;
```

returns

Name
Fred
Jean
Sally

DISTINCT (as explained in my SQL Workshop) forces SQL to remove the duplicates in the answer table.

In this case (and in all cases where a single table is queried) there is no difference between SELECT and SELECT DISTINCTROW. However, if we bring two tables into the query, the two forms can be distinguished.

```
SELECT Name
FROM CLIENTS INNER JOIN ORDERS
ON CLIENTS.ClientID =
ORDERS.ClientID;
```

returns:

Name
Fred
Fred
Fred
Fred
Jean
Jean
Jean
Fred
Fred

which is one record for each order in the ORDERS table because the join between the tables has essentially generated nine records, of which we have asked to see only the name of the customer. By contrast,

```
SELECT DISTINCT Name
FROM CLIENTS INNER JOIN ORDERS
ON CLIENTS.ClientID =
ORDERS.ClientID;
```

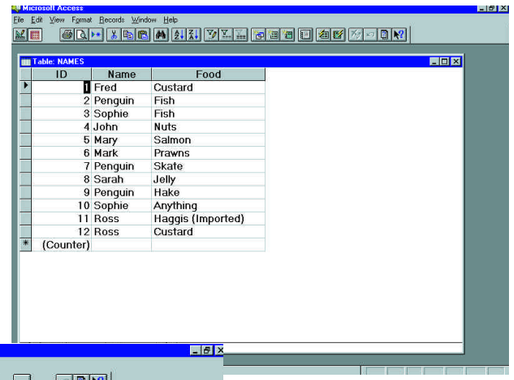
returns only two records:

p276 ➢

Name
Fred
Jean

The SELECT statement on its own returns nine records, then the DISTINCT part removes the duplicates.

The "problem" with this answer table is that it implies that there are only two people placing orders, whereas we know it is three because there are two people called Fred. If we expand the SQL statement to include the location field,



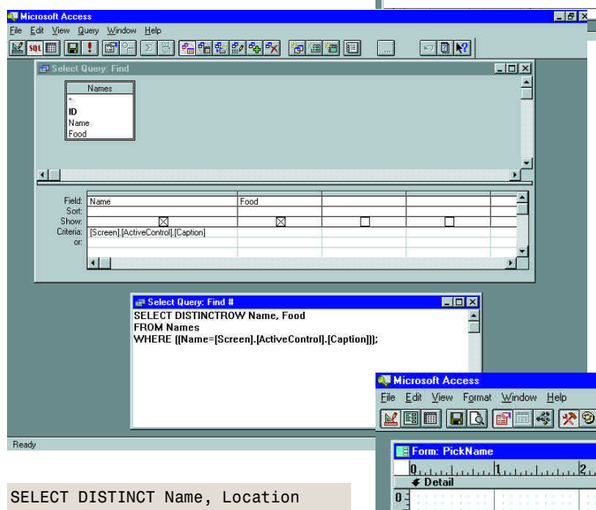


Fig 1 (above) The names used for the button-caption example

Fig 2 (left) The query called Find, as both GUI and SQL

```
SELECT DISTINCT Name, Location
FROM CLIENTS INNER JOIN ORDERS ON
CLIENTS.ClientID = ORDERS.ClientID;
```

then we get:

Name	Location
Fred	Lancaster
Fred	Wellington
Jean	Halifax

DISTINCT is very literal; it returns unique records in the answer table, whether or not they come from unique records in the original table.

I used the term "problem" above, but of course this is only a problem if you don't know what DISTINCT is supposed to do. In fact, DISTINCT is doing exactly what it was designed to do.

My guess is that Microsoft felt that naïve users of a database might not appreciate this level of subtlety. If they saw a single name in an answer table, they would expect that it represented a single person. So the default in Access was set to DISTINCTROW which in this case, as you will by now have guessed, produces a separate record in the answer table for each unique customer who has placed an order.

Thus:

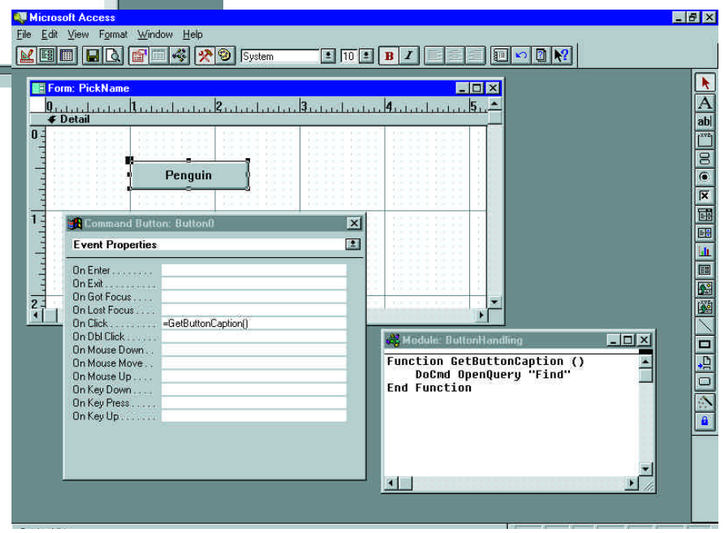


Fig 3 The form called PickName and the associated function which calls the query called Find

```
SELECT DISTINCTROW Name
FROM CLIENTS INNER JOIN ORDERS
ON CLIENTS.ClientID =
ORDERS.ClientID;
```

returns

Name
Fred
Jean
Fred

By the way, these tables and queries are all on our cover-mounted disc as DISTINCT.MDB.

■ I'll leave you with this brainteaser: *If you include the primary key value from CUSTOMERS in all of these queries, does the difference between SELECT DISTINCT and SELECT DISTINCTROW disappear*? As they say in all the best text books, explain your answer!

Hundreds and thousands

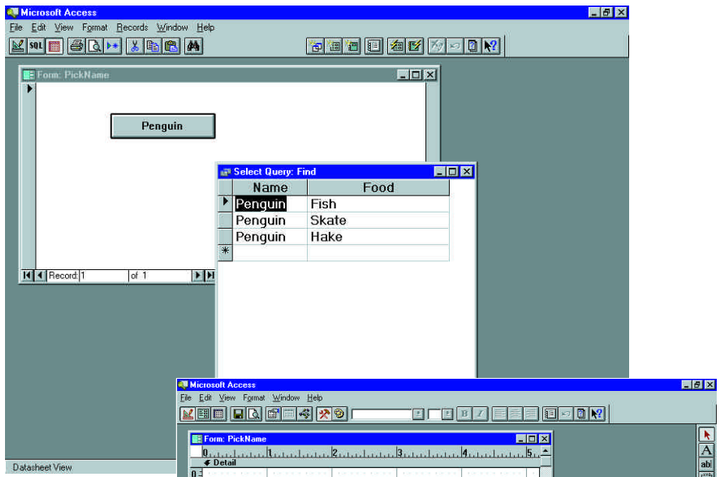
Reader, Glenn Rowe, recently contacted me with an interesting database problem: "*I'm developing an application which stores data about many countries. The data is stored in a table, one field of which stores the country code (GB for Great Britain… and so on). The interface will contain many buttons; one for each country. Each button will have the country code as it's caption and, when pressed, will return the records for that country. (These are, of course, arranged hierarchically on many forms, not all on one!)*

The problem, as far as I can see, is that I'm going to need hundreds/thousands of queries and as many snippets of code. Building it will be a nightmare, as will debugging and maintenance. Surely this process can be centralised in some way?"

The question is a good general one about the way in which code and components can be versatile and non-specific. The obvious solution, at first, is to try to pass the button's caption to the query as a parameter.

As far as I know (and I stand to be corrected) this cannot be done. However, we can achieve exactly the same result by building a query which snatches the caption from the button which has just been pressed. In other words, you can't attach a piece of code to the OnClick event property of a button which says "Run a query and pass to it the caption of this button". Instead, you get the button to run a query and get the query to locate the caption of the button which has just been pressed.

In order to do this, we can make use of an object called "Screen" and one of its properties, "ActiveControl". These are defined in the manual thus: the Screen object refers to a particular Microsoft Access form, report, or control; the ActiveControl Property is used to refer to the control that has the focus. Or, to put that another way, Screen.ActiveControl always points to the control which has the focus.



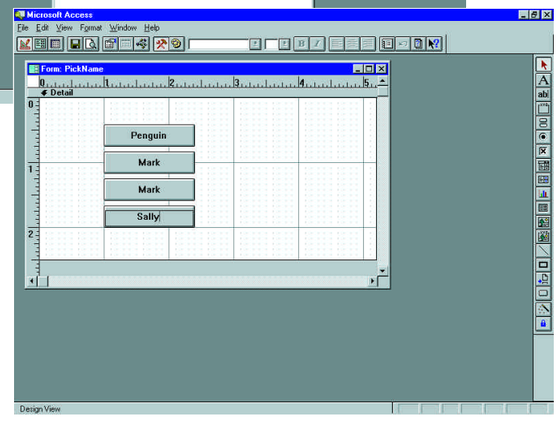


Fig 4 The answer table which appears when the button labelled Penguin is pressed

Fig 5 Cloning the button

Now consider a button on a form; if you press it, that button, by definition, has the focus. If the button runs a query which has Screen.ActiveControl.Caption as its criterion, then Property will return the caption of the button.

The beauty of this scheme is that when you design the query, you don't have to know which button is going to be pressed. You don't even need to know which form the button will be on. As long as the button calls up the query when it is pressed, the query will seek out the button, read its caption, and use that caption as a criterion.

To demonstrate this, I have used people rather than countries. This choice simply reflects the fact that I don't know enough country codes to fill even a sample table.

The table shown in the screenshot Fig 1 contains names of some individuals and the food they like to eat. The query called Find (Fig 2) has, in the criteria line:

```
[Screen].[ActiveControl].[Caption]
```

The form called PickName (Fig 3) has a single button with the caption "Penguin". The OnClick property of this button is set to

```
=GetButtonCaption ( )
```

which is the name of a function. The

p278 ➢

Round and round we go

In the November issue, I published the following correspondence from Paul le Glassick: "*…Incidentally, to get around the lack of an =Round equivalent, I use the Format$ function. This converts numbers to text, but rounds properly as we know it. With a representative sample of nearly 3,000 records, the following nesting of functions gave correct rounding when calculating VAT:*

```
RoundNo = Val(Format$(CCur(Number), "0.00"))
```

where RoundNo is the result and Number is the number or calculation to be rounded."

Paul wanted to know if there was a better way. Simon Hawkins suggested:

```
RoundNo = int (( Number * 100) +1) /100
```

"*Whether this is a faster method than using the Format function, described in the article, I am not sure. Also, the above may need altering to deal with negative amounts. Hope this is of some use.*"

This has the great advantage of elegance. However, when I tested it in the form

```
Function Simon (Number) As Integer
   Simon = Int((Number * 100) + 1) / 100
End Function
```

it returned 1 from 1.49 (which seems right), but 2 from 1.4999 (which seems wrong), and then 2 from 2.49999 (which seems right), and even 2 from 2.499999999 (which is still right, but conflicts oddly with the result from 1.4999!). In other words, it is slightly inconsistent. Or maybe it's my 486 processor. In the form:

```
Function Simon2 (Number) As Double
    Simon2 = Int((Number * 100) + 1) / 100
End Function
```

it returns 1.51 when given 1.5 (which seems wrong).

This reply came from James Talbut: "*I don't like converting things to and from strings. There is an operator in VB that appears to round correctly, and that is the ' \' operator (integer division). Making use of this in a function is simple:*

```
Function Round(dNumber As Double, iNumDigits As Integer) As Double
Dim dFactor As Double
  dFactor = 10 ^ iNumDigits
  Round = ((dNumber * dFactor) \ 1) / dFactor
End Function
```

"This function is simple, quick, and produces the same results as the version using Format$ that you published. Interestingly it gives a different result to that of the ROUND function in Excel. For some reason Round(2.15, 1) does not give 2.2 (as it would in Excel), it gives 2.1 as does the formula you published."

All of these suggestions work, up to a point, but none are perfect (see the form called "Rounding" in Fig 6). So the plot thickens. Anyone got any further thoughts?

By the way, just as we were going to press, Simon came back with: "*OK. Classic case of using my memory instead of looking the code up (and testing it!). The function should read*

```
   TxtOutput =
Int((TxtInput *
100) + .5) / 100
```

"This will round to two decimal places. Sorry about the earlier confusion."

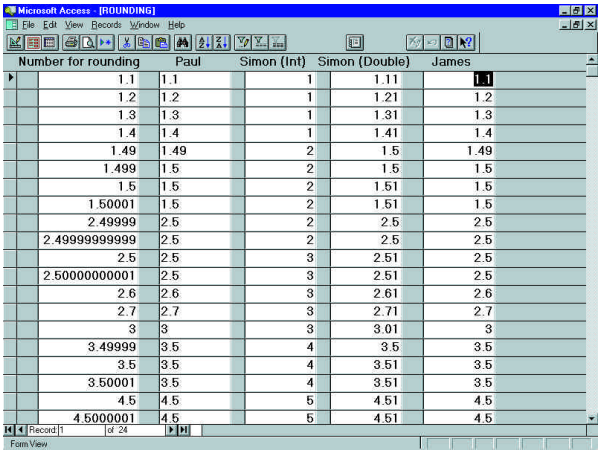


Fig 6 The rounding functions in operation. Note the purely fortuitous grouping of biblical names!

function, also shown in Fig 3, is composed of a single line which simply calls the query.

So, the steps are simple: when you press the button, the function is called; the function runs the query; the query looks for the active control (which is the button), captures its caption, and uses that caption as a criterion. The result is the answer table shown in Fig 4.

To keep it short and sweet, there is no error trapping in the demonstration code. It is also rather crude. For example, unless you close the query, pressing a different button won't re-query the table. However, all of this can be cured by expanding the code in the function from its current, rather minimalistic, one line.

Having answered that question, I realised that, essentially, the solution relied on the fact that the function could be called as soon as the button was pressed. This meant that the solution couldn't be applied to a situation where several selections needed to be made. I know there are manifold ways of handling multiple selections and that combo boxes are often the best solution, but I was intrigued to see if an economical solution (in terms of code and number of queries) could be found while retaining the button-caption-snatching idea.

The form MultipleSelections shows this in operation. A little of the elegant simplicity of the earlier solution is lost but much can be retained. Each button runs a function which places the button's caption in the text box above it. This text box is explicitly named in the function, so each button can be cloned vertically but each separate column of buttons is calling a separate function. Thus, if we made ten selections, we would need ten functions. However, by using Screen.ActiveForm (a close relative of Screen.ActiveControl) in the functions, a degree of flexibility has been retained so that forms which are cloned from this one should also work, as long as the same names are used for the text boxes.

The "Go" button fires a query called (rather imaginatively, I feel) "Find2". The query looks at the text boxes in the form, takes the values from there and runs the query. By using Screen.ActiveForm this query should be versatile enough to work with different forms.

PCW Contacts

Mark Whitehorn welcomes readers' correspondence and ideas for the Databases column at **database@pcw.vnu.co.uk**

p000 ➢

The video **top ten**, again...

A recent query regarding video rentals has prompted some keen response from readers. Mark Whitehorn presents some of your solutions, plus some general tips and teasers.

In the October issue I published a couple of questions from readers. The first concerned a video rental company which wanted to find out the most popular videos rented out over a specific period, essentially the "Top Ten" videos for that period.

Several people sent in further suggestions, including the following from Roger Moran: *"By 'standard' SQL, I assume you are referring to the SQL/92 standard, and explicitly excluding all the various vendor-specific extensions such as the 'TOP' command in Microsoft Access SQL."*

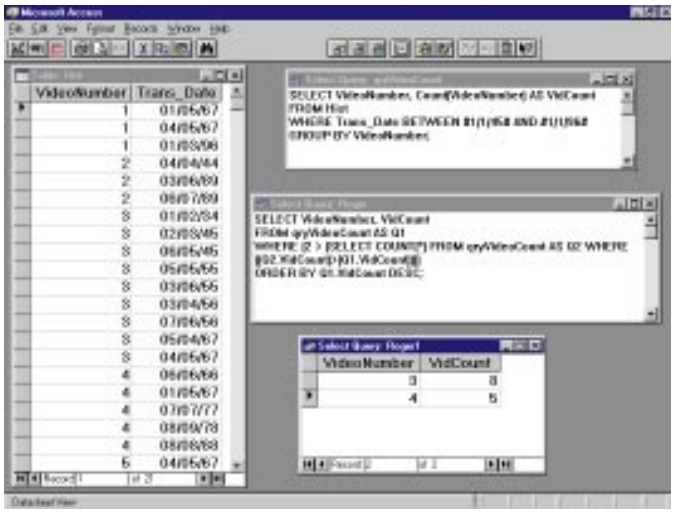
No, but the misunderstanding is my fault for not being accurate in my choice of words. Eamonn Mulvihill's original request was for a solution which worked with Borland's Local-SQL which he said didn't support nested SELECT statements. I should have said something like "lowest-common denominator SQL" rather than "standard". Roger Moran continues: *"Since MS Access seems to be your first choice when describing problems in your column, I will use MS Access SQL syntax in the following, although the SQL is easily ported to any ANSI-compliant SQL/92 dialect.*

"We start with an aggregate query to provide us with a count of transactions per video within a given date range. This is trivial:

```
SELECT VideoNumber,
 Count(VideoNumber) AS VidCount
FROM Hist
GROUPBY VideoNumber
WHERE Trans_Date BETWEEN
 [StartDate] AND [EndDate];"
```

I know it will sound picky, but although the sentiment behind this is quite clear, I think the syntax for Access actually needs

p296 ➢



Here is Roger's solution. The only change I have made is to include the dates into the first SQL statement so that the query runs without putting up dialogue boxes

Dear Santa...

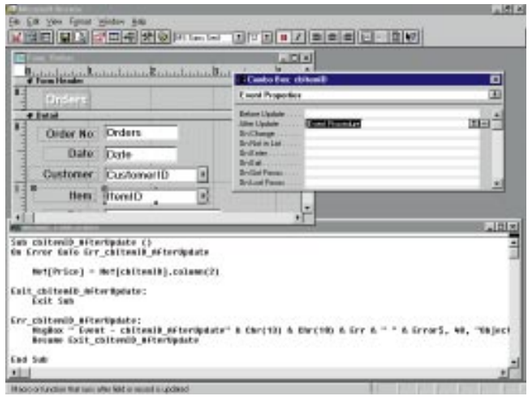
Last year, I asked you to make our current crop of RDBMSs a better match to the relational model, and also for more and better database design tools. I don't know about the rest of the kids, but frankly, I was just a little disappointed. None of the RDBMSs has shown a marked improvement, and the tools situation is still dire.

Sigh. Perhaps I'm asking too much. After all, the vendors don't seem to be interested in what I asked for either, so you must face an uphill struggle in trying to persuade them. This year, I'll try for really achievable requests, but first, a bit of background. Many database professionals are having their serious work disrupted by trivial requests from the management about putting data onto this "web" thing. Now, we all know that the web is just a passing fad. Just like CB radio, once the fuss has died down, I'm sure most people will go back to the telephone. But while it is fashionable, we have to look keen. So, what we need are sensible tools for handling data from databases on the web.

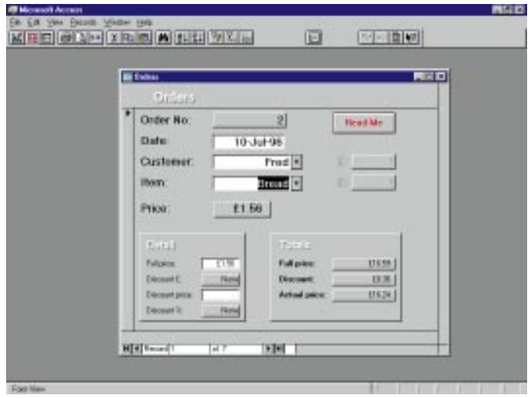
Borland has already made a good start with IntraBuilder, and Microsoft is following with the improved web-publishing abilities of the Office products, but there is still a huge way to go. Neither of the products addresses the problems of signalling between the browser and the database to allow proper transaction control. This is not to say that transaction control is impossible, just that it is currently all left to the programmer. There are other problems as well. With present technology, if the browser sends a query which returns, say, 140 records, those are usually buffered at the web server and sent to the browser in chunks of, say, ten. This is crazy! Every time the user wants to see another ten, there is a pause as the server is contacted and the next ten are sent. This problem isn't going to be solved by a single product; it needs to go right to the standards that these technologies use. If you could wangle your way onto the appropriate standards committee to address this one, we would all be eternally grateful.

By the way, I'd still really like a mouse mat with a penguin on it. I mentioned this last year, but it must have got lost in the sleigh. Perhaps you could check under the seats?

(right) Tony Wall's solution, showing the form and the code behind the combo boxes



(below) Geoff's form, which also totals the orders



to read as:

```
SELECT VideoNumber,
 Count(VideoNumber) AS VidCount
FROM Hist
WHERE Trans_Date BETWEEN StartDate
 AND EndDate
GROUP BY VideoNumber;
```

"In MS Access, we can use this query directly as the datasource for the second stage. In other products which do not support daisy-chaining queries in this fashion, it is a simple matter to convert the above query into an append query which populates a temporary working table, and then use that as the datasource instead.

"Let's assume we have saved the above query with the name [qryVideoCount]. The 'Top Ten' query will look as follows:

```
SELECT VideoNumber, VidCount
FROM qryVideoCount AS Q1
WHERE (10 > (SELECT COUNT(*) FROM
 qryVideoCount AS Q2 WHERE
((Q2.VidCount)>(Q1.VidCount))))
ORDER BY Q1.VidCount DESC;
```

"Here's a basic description of what's going on in the above query. For each record in qryVideoCount (Q1), we find the number of records in qryVideoCount (Q2) whose VidCount is greater than that of the 'current' record in Q1. If that number is less than ten, we know the 'current' record should be included in our 'Top Ten' list. We apply the test to each record in Q1 and the result will be the 10 desired records.

"The aforementioned query works, and is written in standard SQL, but it cannot be considered very efficient, since for each record in qryVideoCount (between 1 and 12,000 in Eamonn Mulvihill's situation), we must execute the sub-query which determines whether the record is to be included in the 'Top Ten' list. In a real world situation, it would probably be quicker to write some procedural code to pull out the ten records from qryVideoCount.

"Hope this information is useful."

Yes, it certainly is. It won't do as a solution for Eamonn's problem because of the nested SELECT, and I agree that it is likely to have speed problems on a large data set. However, Roger is quite right that his is a solution comfortably within SQL-92 and I have great admiration for its elegance!

You can clearly tweak the second SQL statement very easily to alter the number of records that it returns, simply by changing the numeric value in the line:

```
WHERE (10 > (SELECT COUNT(*) FROM
 qryVideoCount AS Q2 WHERE
```

I have changed it to two in the sample database (ROGER.MDB), because the HIST table has so little data in it.

Alistair Logie sent in this solution for FoxPro users faced with a similar problem: *"I read with interest your PCW Databases column about SQL selections of the top ten rented videos. One solution in FoxPro would be to use two SQL statements and a Cursor:*

```
SELECT  VideoNumber,
 COUNT(VideoNumber) AS VidCount ;
 FROM hist ;
 INTO CURSOR mytable ;
 GROUP BY VideoNumber ;
 ORDER BY VidCount DESC

SELECT  VideoNumber,  VidCount ;
 FROM mytable ;
 WHERE RECNO() < 11
```

"I don't know if this is usable on other databases. It does rely on being able to use the absolute record number, RECNO(), as a selection field. The attraction of using a Cursor as a temporary file is that FoxPro creates and erases it itself."

The second question in the October issue covered a more general problem. Suppose you have a set of Customers and Items for sale. Given that each item has a specific, unique price, it is clear that the price fits into the ITEMS table. If each Customer has a unique price for each item, it is equally clear that the price needs to go into a separate table which lists each customer, item and the unique price for that key value.

What if the situation is somewhere in between, where most prices are standard but there are some exceptions? I produced a solution, but asked for suggestions, improvements or comments. The first comes from Tony Wall, and his database is on the CD as TONYWALL.MDB.

"I have enclosed an alternative solution to the Customer & Prices problem from the October issue. Whether it will work efficiently with the 400 customers and 300+ products is another matter.

"In the first instance I took your Default List and Exception List and used these to create a union query as follows:

```
 SELECT *
 FROM [PriceList] as PL
 WHERE NOT EXISTS
 (SELECT *
 FROM [ExceptionList] AS EL
 WHERE PL.[CustomerID] =
EL.[CustomerID] and  PL.[ItemID]
=EL.[ItemID]);
 UNION SELECT *
 FROM [ExceptionList];
```

"This is used as the basis for the combo box data source on the orders form for retrieving the correct price."

Tony then sent another mail message, saying that the performance was terrible with a larger data set. His suggested improvement is well worth looking at, and is in TONY2.MDB. Geoff Wyss sent in another; see GEOFWYSS.MDB.

"In the October 96 issue of PCW, you posed a problem about merging prices from two sources to make up an Order from a default price list and an exceptions list. The attached database file shows two possible solutions which might be of interest, one using a Union query, the other using an IIF statement. Click on form Orders2. There is a ReadMe button on the form which gives an explanation."

For this month's problem, I have held off producing a solution myself. This is mainly due to cowardice, since this problem brings us once again into the holy wars of "strict relational" vs. expediency. I tend towards a purist view, but in this case I can see reasons for a more heretical solution. Before committing myself to be burned at the stake, what do you think? Let's try not to get too wound up about this one. It is Christmas, after all.

"I am a teacher, and recently used Microsoft Access 2.0 to produce a database for recording and reporting on pupils' performance in tests. To keep the database as flexible as possible in use, I store test results in a single table with four fields: each record in the Scores table includes an identifier for the pupil, an identifier for the test, the percentage score achieved by the pupil in the test, and the position of the pupil out of his/her teaching group. Records are added to the table as data are entered by the class teacher. (The pupil and test identifiers together make up the primary key, as each pupil appears in the table many times, as does each test, but each pupil can only take a given test once.)

"It is useful to have the computer calculate the position of the pupil automatically, once the data for a full teaching group has been entered, but I have not found a function in Access to do this easily. The pupils' teaching groups are recorded in a separate table. It is straightforward to produce a query listing for a particular test, the pupils in one teaching group and their percentage scores, and this query can be sorted appropriately. But then to work out the position of the pupils within the group and store this in the appropriate field, I have resorted to a complex sequence of operations involving creating and deleting temporary tables.

"Using this method to calculate pupils' positions slows down the program significantly and, because of the use of temporary tables, has led to problems when two teachers using the network attempt to enter test scores at the same time. Is there a direct way to fill in the 'position' field in the Scores table? I have tried different techniques, but have not found a wholly satisfactory solution." Andy

Andy's test database (not containing real data about real children) is on the CD as ANDY.MDB.

Teasers

A couple of teasers from Jeff Jenkins:

1. Strange database design

A database I was working on stored amounts as positive numbers and had a separate field to indicate the sign of the number. This field had values of one or two to indicate positive or negative (don't ask why, I didn't design the database). The problem was how to extract the amounts as signed numbers. The solution I came up with was a mathematical one, selecting "amount*(3-(sign*2))". This works as "(3-(sign*2))" evaluates to 1 or -1. A bit of a rigmarole just to get a signed number out.

2. Summing only the positives

I needed to provide a total of all the positive numbers in a particular field in the database (e.g. all the accounts with a credit balance). I couldn't find an sql command to do this, so I resorted to another mathematical solution: "`SUM((amount+ABS(amount))/2)`"

Adding the absolute (unsigned) value of a number to itself comes out to zero for negative numbers, and dividing by two negates the effect of adding a number to itself, so the sum works. You can also sum up all the negative numbers with:

"`SUM(amount-(ABS(amount))/2)`"

Incidentally, the first one also works as

"`SUM((ABS(amount)+amount)/2)`"

but the negative version

"`SUM((ABS(amount)-amount)/2)`"

gives a positive total of all the negative numbers.

PCW Contacts

Mark Whitehorn welcomes readers' correspondence and ideas for the Databases column. He's on **m.whitehorn@dundee.ac.uk**

Modelling job

Where did you put that data? Where did you put that file? Mark Whitehorn looks at the pros and cons of database modelling.

Last month, we started to look at three "database models" you can employ:

1. Everything on a stand-alone PC;
2. PC front end with data on file server;
3. Client-server using a database server as the back end.

In this context it is useful to think of a database as having four different parts:

1. User interface section;
2. Data processing engine;
3. Conflict resolution section (to deal with conflicts introduced by multiple users accessing the data at the same time);
4. The data itself.

We looked, last month, at the pros and cons of the first model. As I wrote at the time, I will give size estimates where appropriate but please don't take them as gospel. (See last month's *Hands On Databases* for other qualifications).

PC front end with data on file server

If you have a need for multiple users to access the same data, then it is not beyond the realms of possibility that you already have a network. Given a network, you have the option of moving to our second database model.

In this model, much is left the same as before. You would still run Access, Paradox, or the RDBMS of your choice on your PC so the data processing engine bit of the database stays there.

Only two things change. One is that the data files are moved to a file server and the second is that the individual RDBMSs running on the individual PCs need to communicate with each other. They need to do this in order to resolve the multitude of potential conflicts which suddenly arise when more than one person accesses the same data simultaneously.

Now is not the time to go into all such conflicts but consider a simple example. You and I both work for the same company and we are trying to update the company's customer records. I open up the record for A Smith to increase his credit rating from £2,000 to £3,000. While I am doing so, you delete his record. What happens to his record when I finish editing it and send it back to the file server?

The answer to this question depends on the RDBMS you are using. Access, for instance, maintains a lock file in the same directory as the data file and this is used to store information about who is doing what at a particular time. Thus, if I had opened the record to update it before you tried to delete it you would receive a message saying that the record was in use by "Mark" and that you wouldn't be able to update the record until I had finished with it. Other RDBMSs use other mechanisms, some less efficient then others, for dealing with these potential conflicts.

The important point is that with this database model the control of the user interface and the data processing engine remain on the PC while the data and conflict resolution are moved to the file

Delphi Programming for Dummies

by Neil J. Rubenking

Delphi is a multi-purpose programming tool and is commonly used for the generation of user interfaces to databases. This book is one of a series and if you've seen any of the "** for Dummies*" books you'll already have a good idea of what this one is like.

It has lots of diagrams, some very silly cartoons and lots of icons in the margins to identify Technical Stuff, Warnings, Tips and so on. It is also written in a style which doesn't display the correct reverence for what is a serious programming tool — which is just fine by me. It's not that I don't respect Delphi but books without a sense of humour can be really tedious. This one is fun.

The database section is pretty skimpy, comprising a mere 22 of the 376 pages. It would be impossible to recommend this book to a database-naïve user who wanted to learn how to create databases in Delphi but that description doesn't fit most of the readers of this column. If you are happy that you understand the database end of things, this is an excellent introduction to Delphi.

One of the examples used in the book is a function called Hailstone. Given a seed number, this function generates a series of numbers which bounce up and down "like hailstones in storm winds" before converging on the number 1. Apparently there is no way of either predicting how long a given seed number will take to reach 1, or proving that all numbers will eventually reach 1. I was so intrigued that I modified it to run in Access and it is included in the main part of my column for your amusement (see *Hailstones*).

By the way, I saw one of this series in a bookshop in France: "*Windows pour les Nuls*"; a gift of a translation for any of the series connected with databases!

■ *Delphi Programming for Dummies*. IDG Books £18.99 (ISBN 1-56884-200-7).

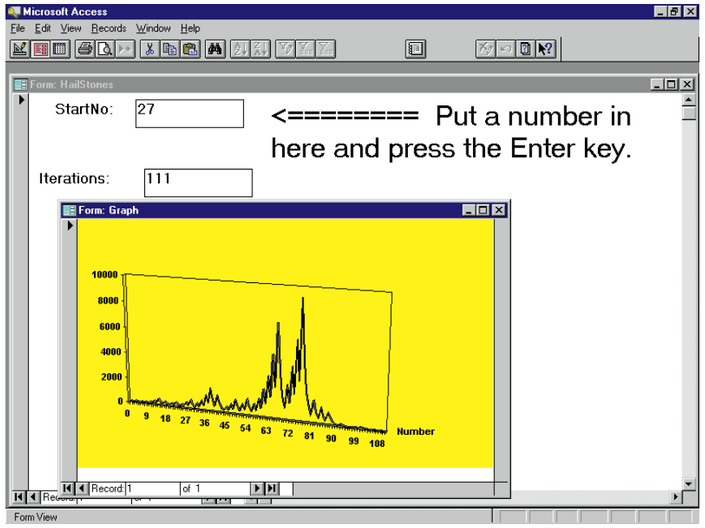


Fig 1 **The "Hailstones" algorithm in action**

server.

The big advantage of this model is that it provides multi-user access to the same data at relatively low cost. The big disadvantage is that the model is inefficient in two main ways. Firstly, it tends to load the network, soaking up bandwidth like a sponge. Remember that the data is at one end of the wire and the processing is at the other. Every time you query the data, it has to be moved to the client since that is where it is crunched. In a badly designed system this can mean that every query against a 100Mb table requires the entire table to be shipped to the client. Intelligent indexing can reduce this considerably (since the indexes can be shipped for searching and only the relevant records sent out to the client) but in practice, the effectiveness of this depends on the particular RDBMS.

Secondly, the processing is at the client end, so each client needs sufficient resources to cope with the data. If you decide that an increase in the database size warrants an increase in memory of 16Mb, you will need to add that to all the clients: given ten clients, that's 160Mb.

These restrictions mean that the number of simultaneous clients and the size of the data are relatively restricted. Think in terms of ten clients and 1Gb.

Client server model

This model is simply a modification of the previous one. The user interface stays on the PC: the data, the processing and the conflict resolution moves to a server. This is typically not a file server but a server dedicated to running applications like RDBMSs, hence they are generically known as application servers. Machines which are dedicated to running RDBMSs are often called database servers or SQL servers.

But wait. Why go to the expense of a dedicated database server when you have that nice NetWare 3.x file server already in place? Can't you run a database engine on that as well? The answer is that you can, but you probably don't want to. The reason lies in the NOS (network operating system).

NOSs like NetWare 3.x are optimised for File+Print. Application servers run a NOS which is optimised for running multi-user applications. This doesn't mean that you can't do it, just that performance will be compromised, so a dedicated database server is generally considered to be better. If you really want to run one server for both application and File+Print, then NT is probably better than NetWare, all things considered.

The Client-Server model is typically *not* limited by bandwidth. Since the processing and data are now snuggled together in one place, queries no longer mean that masses of data have to move across the network. Instead when the GUI, running on the client, is used to construct a question only an SQL description of that query is shipped across the network to the server.

This SQL will typically be a very short ASCII string. The database engine on the server processes the query and simply sends the answer (rather than the entire table) to the client. Conflict resolution is also handled centrally, with associated benefits in terms of speed and sophistication. Centralising the processing means that the whole system is easier and usually cheaper to update. If the database slows down you can throw hardware (memory and processors) at the server. You don't have to add it to the clients because they are simply handling the user interface.

On the ticklish subject of size, this model will typically support as many clients and as much data as you can afford. To put that another way, consider this model seriously if you think you will need more than ten clients or >1Gb of data in the foreseeable future.

Hailstones

The hailstones algorithm is very simple and most easily expressed in pseudo code.

```
Start with a seed number
Repeat
        If the number is even,
divide by two
       Else multiply by 3 and add 1
Until number = 1
```

I can offer no justification for implementing this in Access except that it is fun and that the behaviour is really wacky. Give it a number like 26 and it takes a mere ten iterations to get down to one. But 27 takes a monstrous 111 iterations, and 28 a more reasonable 18. A copy of this is in the MDB file on our cover-mounted disk this month. (See also, the *Delphi Programming for Dummies* panel on p285).

More on meters

The meter problem (which was discussed earlier this year) resulted in my publishing, on our *PCW* cover disk, a database with several of the solutions and a kit which allowed you to test any other solutions against the published ones.

I asked only for solutions which were genuinely faster. None have arrived but Paul Bloomfield, a supplier of one of the original solutions, contacted me to say that he had got very different answers from those published. It turned out that he hadn't got the *PCW* cover disk, so had built his own database for test purposes. I sent him the one I had used and this is his reply:

"Our speed differences are down to one factor: the data. My test database had 2,000 records from 200 meters (i.e. average ten readings per meter), whereas I see yours had only three meters and so 666 per meter, so the query is doing 66 times the work!

Perhaps there are lessons to be learnt here in problems of scaling. There is no such thing as good SQL per se*, the query must be written to fit the data, or the data structured to fit the query!"*

I agree. It is un-nerving to realise how many factors we need to consider when working with databases. Still, it keeps us all in employment!

Case-sensitive joins

Andrzej Glowinski writes: "I have recently started using MS ACCESS (Win 3.1) at work, developing applications in the areas of large medical (clinical) information resources. We also use an Oracle server and my systems work fine using this but as soon as I try to build stand-alone versions all hell breaks loose. This is because some ACCESS designer/implementer, in their wisdom, has forced all internal joins to be case insensitive — and there is NO MECHANISM for altering this behaviour!"

I replied to this as follows: "I hate to be contentious, but Access does provide a mechanism which allows joins to be case sensitive.

Assume that we have two tables: NAMES and ORDERS. Each table has a field called NAME which contains case sensitive data. We thus have two fields:

```
NAMES.Name
ORDERS.Name
```

which need to be joined.

Create a query containing both tables, join them within the query (not in the relationships editor) and run the query. Precisely as you suggest, the data in the tables is joined in a case insensitive manner.

Now add a field to the query:

```
CaseCompare: StrComp([NAMES].
[Name],[ORDERS].[Name],0)
```

This uses the function StrComp which can be rendered case sensitive by setting the third argument to be 0.

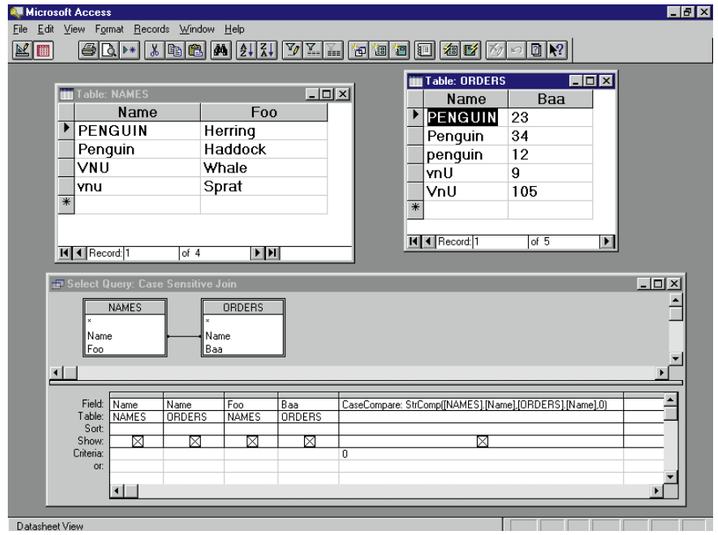
The function returns 0 if the strings match in case as well as letter order, so if you set the Criteria for that field to be 0 and re-run the query, the join is now case sensitive. The SQL version reads as:

```
SELECT DISTINCTROW NAMES.Name,
ORDERS.Name, NAMES.Foo, ORDERS.
Baa, StrComp([NAMES].[Name],
[ORDERS].[Name],0) AS CaseCompare
FROM NAMES INNER JOIN ORDERS ON
NAMES.Name = ORDERS.Name
WHERE (((StrComp([NAMES].[Name],
[ORDERS].[Name],0))=0));
```

I am aware that this doesn't allow you to sort case-sensitive material, although it would almost certainly be possible to write a function to do this. Rather more interestingly, it doesn't allow you to apply referential integrity or set a primary key on material which is case-sensitive (since Access treats 'PENGUIN' and 'Penguin' as equivalent in a primary key). However, it does at least allow you to perform the join." (A copy of this is available in the MDB file on the *PCW* cover-mounted disk).

To which I received Andrew's reply: *"Thanks for your message about ACCESS. We have explored the mechanism you propose quite extensively, both on local (native) ACCESS tables and on attached ones, primarily ORACLE. The performance hit is just unacceptable — I'm talking of joins across tables with 200,000++ items in them, so the total loss of indexing (which, effectively, is what you get) is very significant. You can mess around with the order of execution of the query but it is just too unpredictable to be of much use."*

I can't argue with any of this. Indeed, depending on how large the records are, this database may be getting to the size where a move to another model is inevitable. One question is, do any other PC-based RDBMSs have a better way of supporting



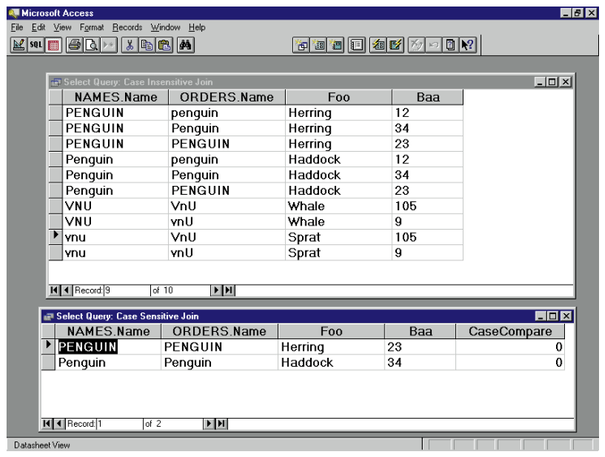


Fig 2 (above) The tables used to demonstrate the case sensitive join, and the query itself

Fig 3 (left) The results of a case insensitive (top) and case sensitive join (bottom)

PCW Contacts

Mark Whitehorn welcomes readers' correspondence and ideas for the Databases column. He's on **m.whitehorn@dundee.ac.uk**

One is not enough

A reader worries that as the work gets too big for his company's current database system to handle, which way now? Mark Whitehorn is on hand to dispense this, and other, advice.

I would like a discussion of the relative merits of a client/server database and a networked database application in which there is a server. When working on database applications, I am always considering small standalone systems for one PC. Most of our work so far has been in-house and thus manageable. As time goes by and we are involved in increasingly big projects, I am beginning to worry that the one-PC approach will fall on its face.

I believe I have two options:
1). Put my data on a big server running something like NT and run multiple copies of my Paradox application all pointing at the tables on the server. Then let Paradox handle the problems. I realise I'd have to be careful about record locking and the like.
2). Up-size my data to a client /server application using something like Interbase running on a server (UNIX?) and do it all through SQL, although my knowledge here is very hazy."

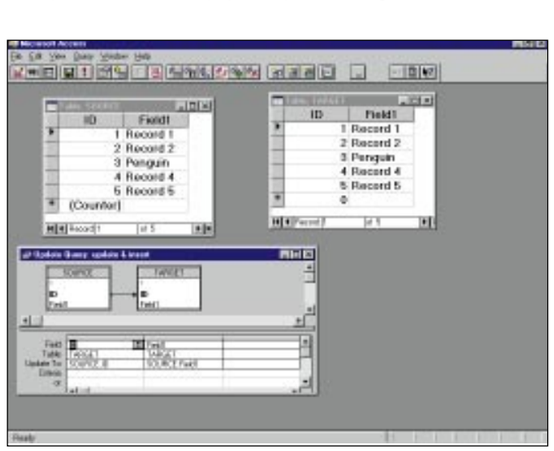
Alasdair Macdonald

I received this email and it seems a broad enough question, to warrant some expansion. After all, it is one of the biggest decisions that you are likely to make, and is an area where mistakes are both common and expensive. There are essentially four database models you can employ:
1). Everything on a standalone PC.
2). PC front-end — data on file server.
3). Client server using a database server as the back end.
4). Mainframe

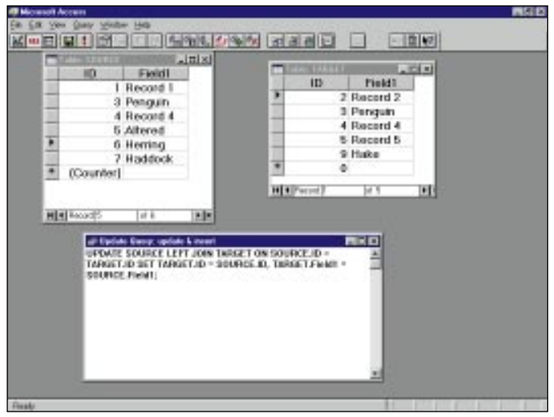
The fourth seems inappropriate for discussion in a PC magazine, so we'll ignore it. What I can do here is to outline the strengths and weaknesses of each of the other three, together with approximate performance and size estimates.

This is a complex area. The decision to choose one of these alternatives will be based upon the interaction of many factors, including response time required, number of users, file size, file number, data size, available resources (including hardware, software and money) and type of data access required (read only, read write).

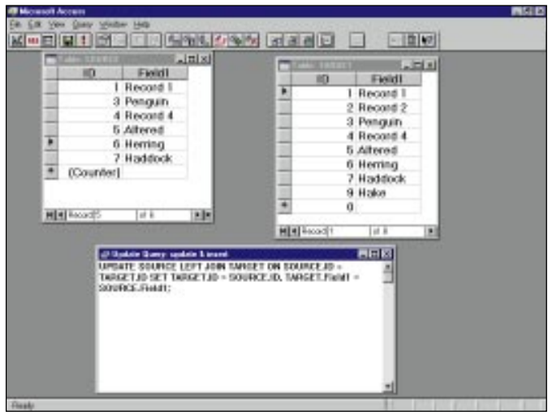
These factors interact in complex ways. Suppose that your system definition makes multi-user access to the data essential: you can instantly rule out a standalone PC. If the number of users is guaranteed to be small (say, three), then on a given hardware



(Fig 1, top right) **The update query described by Paul [see page 276]. The two tables are the same**



(Fig 2, right) **Here I have made changes to both of the tables, but have yet to run the query. (I have also altered the view of the query to show it in SQL)**



(Fig 3) **The state of the tables after the query has been run**

platform you could allow those users to access a greater volume of data than if there were 50 of them. If the number of users did then increase, the system might still work, but the response time would drop.

To make matters worse, the interactions between these factors are often non-linear: for example, doubling the number of users on a given system might have little impact on response time. Doubling it again might bring the same system to its knees.

It's easier not to give any actual figures, but this is likely to leave you gnashing your teeth and wondering "What exactly counts as 'lots'? Three? Twenty five? Five hundred?" On the other hand, if I do quote hard figures, like saying that you shouldn't consider using a standalone PC for more than 1Gb of data, there'll be someone out there happily using a 200MHz P6 with 1024Mb RAM to access 1.5 Gb.

I will quote figures because it seems far more useful to do so, but please just regard them as general figures from which to start discussions. Please don't take them as gospel, and please don't build your entire database strategy around them alone.

Work alone on a standalone

The simplest database model is to install everything on a standalone PC. You use an RDBMS like Access, Paradox, dBase and FoxPro to manipulate the data.

Only one person can use it at a time, and I wouldn't use this sort of system for more than about 1Gb. Factors which affect this figure are the hardware (more memory equals larger data files) and the manner in which the data is used. For example, if it's rarely updated, then it can be heavily indexed and queries should run against it reasonably rapidly. If the data is constantly updated, the indices will slow down the updating, and yet removing them will slow down the querying! In a nutshell, if the data is rarely updated, heavily indexed, and you have very impressive hardware, you can go above this limit. With a 286 with 640Kb, don't even think about it.

The major advantage of this sort of system is that it is cheap, and easy to manage. A database can be thought of as four different parts:
1). User interface section.
2). Data processing engine.
3). Conflict resolution section (to deal with conflicts introduced by multiple users accessing the data at the same time).
4). The data itself.

In a standalone PC-based database there is only one user, so the conflict resolution section isn't required and the other three appear as a single, seamless entity to the user. Simple. In fact, it is *so* simple, why would you ever want to go to anything more complex?

One of the major reasons for moving to a more complex database model is that this one cannot handle multiple users. For one thing, there is only one keyboard, so we can expect fist-fights if we try for multi-user. Also, this model doesn't allow for conflicts between the requirements of different users to be resolved. If you need more than one user to access the data at the same time, it is time to split up the components described above and partition them between different machines. This leads us to the second database model, which we'll cover in the next issue.

Target, aim, fire!

"Suppose we have one table, called Source, containing new records, and another table called Target. We wish to insert into Target the records in Source that are not already in Target, and update those records which are already in both tables to

p276 ➢

Database Systems
by Paul Beynon-Davies

This book looked promising. Many of the subjects covered are of interest, and the style, while a little formal for my tastes, is perfectly respectable. However, reading it in more depth reveals a series of unnerving flaws. For a start, the book is heavily cross-referenced. I like cross-references, but I do like the references to point to the correct place. Far too many here do not. Exactly half of the cross-references in chapter six, for example, are incorrect. With a failure rate like that, they are too frustrating to use. And it is not only the cross-references that are flawed.

The same chapter covers SQL and the author demonstrates retrieval, ordering and grouping for which he uses a base table, eight SQL samples and eight answer tables. Amazingly, three out of eight answer tables are incorrect, a state of affairs likely to induce severe confusion in a novice reader.

This book is aimed at students, but cannot be recommended to them or anyone else, which is sad because in many ways it's a fine book. It is simply crying out for a careful revision. Hopefully the next edition will be improved, but steer clear of this one.

MacMillan Press ISBN 0-333-63667-8. £19.99

the values of the records as in Source.

Instinctively, programmers will achieve this by two queries:

1). Update

2). Insert

In Access it can be achieved with a single Update query with a LEFT JOIN.

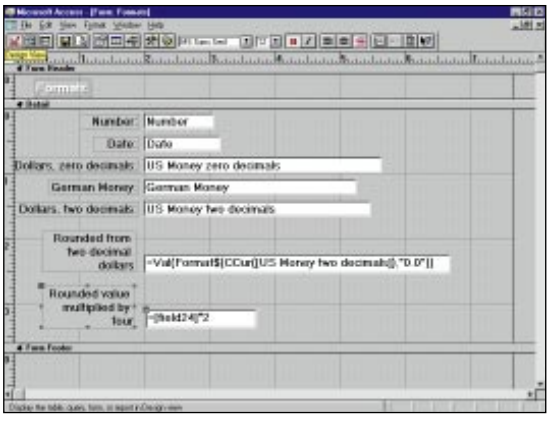
```
UPDATE SOURCE
LEFT JOIN TARGET ON
SOURCE.ID = TARGET.ID
SET TARGET.ID = SOURCE.ID,
TARGET.Field1 = SOURCE.Field1;
```

Note that both tables have fields called ID and Field1, and both are of the same data dimensions in each table. Remember that if ID is a counter field in Source, it must be a long integer in Target. This works, since Access matches those records in Source which are not in Target with a Null Record in Target which can then be updated. From an SQL point of view, this technique may not work in other DBMS but is jolly useful in Access 2.0 and Access 7.0.

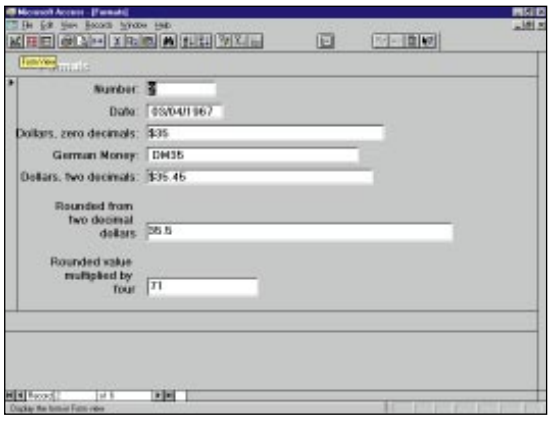
I enclose a sample Access 2.0 database [on the cover disk as pdbdemo.mdb] which demonstrates this. To see it in action, play around with appending/changing records in Source and then run the query to see the effects in Target. The uses of this technique are numerous, and variations on the query to supply selection criteria make it powerful."

Peter Blackburn

It is worth noting that this query will not delete records from Target which have been deleted from Source. This is not a criticism. If it did delete those records, it would effectively be replacing Target with a copy of Source. It is simply a characteristic of this type of query. I can't help feeling that this might help in the solution of last month's problem concerning Mark Squire's problem about Customers and Items.



(Fig 4, right top) Here I have used some of the formats suggested by Paul, and his rounding mechanism



(Fig 5, right bottom) And here is how the data appears. Notice that the value shown by the rounding mechanism is still a number; it can be mathematically manipulated as shown in the lowest "text" box. This form is also in the sample database on the cover disk

Currency codes: help wanted

"Your July column covered the formatting options for dates in Access.

It is often overlooked that the Format property in Table, Query, Form and Report design is not restricted to just those formats on the list. The variety of codes available is the same as those used in Excel's Format, Cells, Number dialogue box.

Thus, a code of dd/mm/yyyy will show dates with century, $#,##0 will show amounts with dollar signs, "DM"#,##0 will show amounts in deutschmarks. This latter is especially useful since Access picks up the default currency format from Control Panel. It can then be overridden to show different currencies on a single Form, etc.

The ability to override the currency format has an additional benefit. Since currency fields are held to a fixed four decimal places internally, they are likely to be more precise than single or double numbers which can give puzzling rounding errors. As you know, Access has no exact equivalent to the =Round function in Excel. Currency fields used for other than currency can be formatted as #,##0, or #,##0.00 if two decimal places are required.

Incidentally, to get round the lack of an =Round equivalent, I use the Format$ function. This converts numbers to text, but rounds properly as we know it. With a representative sample of nearly 3,000 records, the following nesting of functions gave correct rounding when calculating VAT:

```
RoundNo = Val(Format$(CCur
(Number), "0.00"))
```

where RoundNo is the result and Number is the number or calculation to be rounded. Do you know of a better way of doing this?"

Paul le Gassick

The answer is that I don't. Anyone else? (See Figs 1-5.)

PCW Contacts

Mark Whitehorn welcomes readers' correspondence and ideas for the Databases column. He's on **m.whitehorn@dundee.ac.uk**

Blood, sweat and **coding**

Mark Whitehorn reports from Database Expo, where he has been judging a programming application contest. Plus, two ticklish problems, taped.

I have just returned from Database Expo, which was run as one of the IT Expo group of events. It's held in Birmingham at the end of June and if you missed it this year, pencil it in for next because it was great. The exhibition organisers also ran a RAD (Rapid Application Development) race as part of the exhibition. The rules are simple. A charity in need of a database is selected (in this case NACRO — National Association for the Care and Resettlement of Offenders). The charity submits a set of requirements from which a formal specification is drawn up. Teams consisting of up to two programmers are given two days to develop an application which meets the specification.

I was asked to be one of the judges and, along with the others, was keen to make the application development in the race as close to reality as possible. So we decided to allow the contestants to use not only any commercial software which took their fancy, but also any and all toolboxes, commercial or otherwise. Although other races of this kind tend to restrict contestants to shrink-wrap development tools only, we felt the open approach was by far the more realistic. How many good developers have you met who don't have their own toolboxes? Additionally, in these object-orientated days, the extensibility of a development tool is a major consideration.

Secondly, we decided that we would announce a change to the specification during the morning of the second day. After all, have you ever worked on a specification which didn't change during development? We felt that this would favour teams and tools which were adaptable: a highly desirable trait in both. After some deliberation, we decided to warn the contestants at the start of the competition that this "Judges' googlie" was coming (mainly through fear of physical violence if we bowled it to them unannounced!).

The competition went well, and was won by the Borland/Dunstan Thomas team. The tool was Delphi and the team consisted of Jason Vokes and Colin Ridgewell. Dunstan Thomas is one of Borland's Client/Server Partners based in Portsmouth which specialises in developing client/server and internet business systems.

One company, notable by its absence from the contestants, was Magic. This is a company which, over the years, has built a comprehensive advertising campaign around winning races of this type. What made its absence all the more apparent was that it had initially been a keen supporter of this particular race. In May, Graham Young, marketing manager for Magic software had said, "We believe that the professional software development industry needs an attractive one-stop shop for showcasing the latest technologies and products. With Blenheim's (the exhibition organiser) backing, the RAD race is well-positioned to become an important milestone in the IT calendar."

With about one week to go before the race, Magic announced that it had decided not to enter a team. The reason given was that Magic felt it had already demonstrated its superiority in this field. Graham Young told me afterwards: "We've already thrashed Delphi on many occasions in the past." While this is perfectly true, we are talking about a different competition, here, with a different target application and a unique set of rules (including the unusual "open toolbox" item).

One is left to ponder whether its past victories were the only reason for Magic's non-participation? For a start, in a rapidly evolving field like database development tools, superiority needs to be regularly demonstrated. Last quarter's victory is as stale as yesterday's news; and for a company to refuse what might perhaps be an easy victory (with its associated positive publicity), may be laudable but is unlikely to impress the shareholders.

Whatever the reason, I really hope that Magic takes part next year. The more teams that take part, the more impressive the win — whoever gets it — and it could be you. Why not talk to your boss about entering yourself and a colleague as a team? We can promise you two days of sweat, blood and coding. What better break could there be from the daily grind?

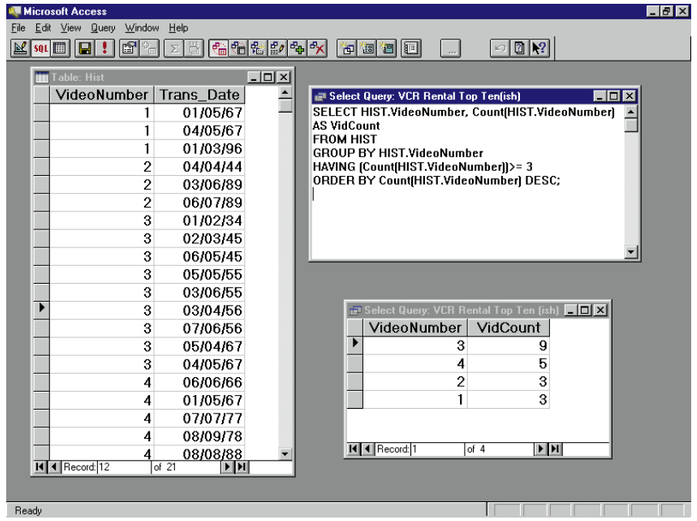


Fig 1 **Since the data files are small, this query is finding only those videos which have been rented out three or more times, but it illustrates the general idea**

A rental problem, taped

"*I am nearing completion of a program to handle videotape rental. Each transaction is written to a history table to provide flexible reporting.*

The table HISTORY contains information about videos and customers but the important fields are Video_no and Trans_Date. Using Video_No and the function COUNT, you can produce a list giving Video_no and the total number of times that video has been hired — essentially a top ten list. If you bring Trans_Date into the equation you can produce a top ten for a specific period, say the last two weeks, which is far more useful.

Unfortunately though, this is not a true top ten list. SQL will return a set containing a record for each video on the system (in practice this is between one and 12,000 records). Is there a way to return just the first ten records? I'm no SQL wizard and in my experience the answer is definitely 'no'. Perhaps you know better?

Here is my SQL (cut to a minimum):

```
SELECT HIST.VideoNumber,
 count(HIST.VideoNumber) as
 HIST.VidCount
FROM ":DBVIDEO:HISTORY" HIST
GROUP BY HIST.videonumber
ORDER by HIST.VidCount DESC
```

I am using Borland Delphi with Paradox tables and Borland's Local-SQL. I'm not looking for an application-specific solution but the low-end SQL implementations often omit features that ORACLE and GUPTA users take for granted. Local-SQL does not support nested SELECT statements."

Eamonn Mulvihill

The good news is that you are correct. My understanding is also that it can't be done with "standard" SQL. The bad news is, of course, that being correct doesn't help you to solve your problem: a bit of a video nasty. Given that what you ask is impossible in Standard SQL, any answer I give is going to be more or less unsatisfactory; but it may be helpful, particularly to other readers. An Access derivation of your SQL would be:

```
SELECT VideoNumber,
 Count(VideoNumber) AS VidCount
FROM HIST
GROUP BY VideoNumber
ORDER BY Count(VideoNumber) DESC;
```

One approach to the problem would be to set a value for Count(VideoNumber) above which you want to see the video. For example, you might know that the top-selling (renting) ones are often taken out, say, 12 times or more. So you could use the form:

```
SELECT VideoNumber,
 Count(VideoNumber) AS VidCount
FROM HIST
GROUP BY VideoNumber
HAVING (Count(VideoNumber))> =12
ORDER BY Count(VideoNumber) DESC;
```

This lists only those videos which have been rented out more than 12 times (Fig 1). I know that this won't necessarily give you exactly ten videos as the answer, since fewer or more may fit this criterion, but you might, with some practice, be able to get approximately the answer you want. For Access freaks, there is a sample table and query in this month's .MDB file on our cover-mounted CD-ROM.

The variation is an Access-specific variation which will give the top 10.

```
SELECT Top 10 VideoNumber,
 Count(VideoNumber) AS VidCount
FROM HIST
GROUP BY VideoNumber
ORDER BY Count(VideoNumber) DESC;
```

As an aside, it is tempting to hope that these SQL statements will execute faster than one which returns usage counts for all the videos. However, whether ten or 10,000 records are returned in the answer table, the entire base table still has to be scanned in order to provide the answer.

An eggsacting problem

"*A client runs a medium-sized food wholesale business. He has asked me to design a system in which each customer can have certain products at different prices. For instance, one client might negotiate a*

p280 ➢

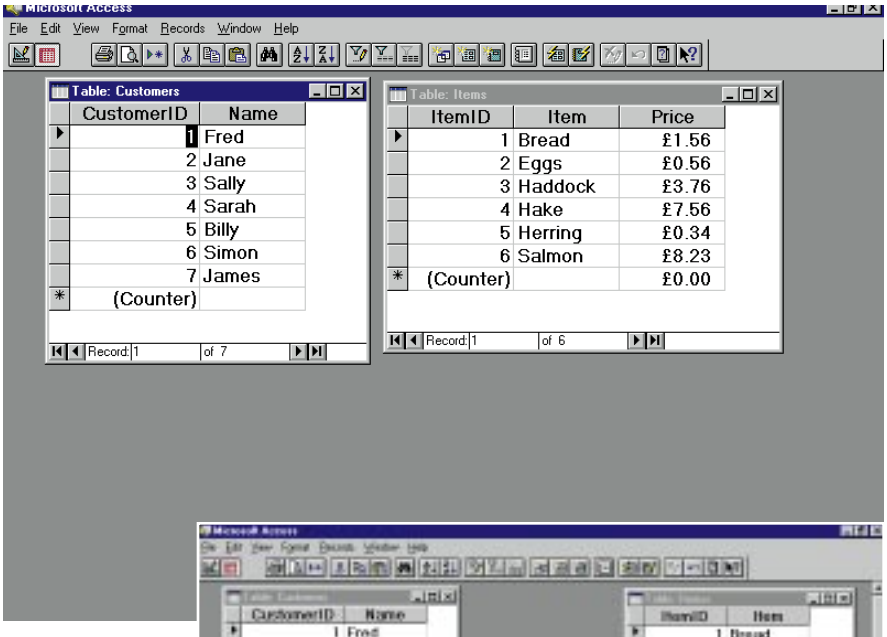


Fig 2 (above) Two sample tables for the second problem. Note, seven customers and six items

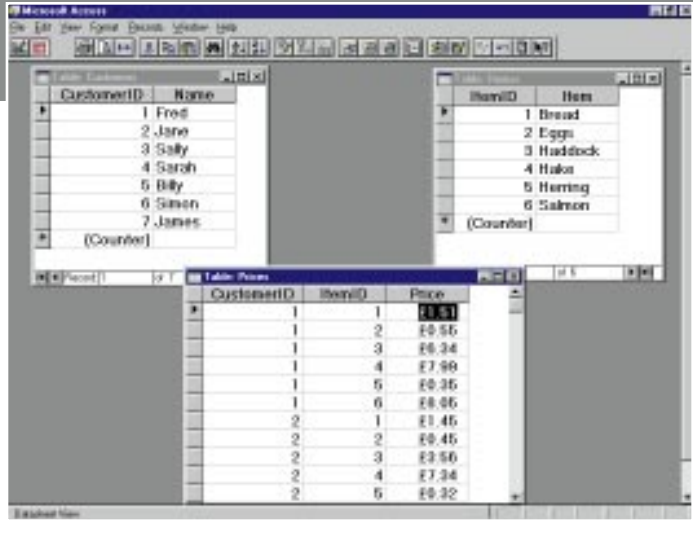


Fig 3 (right) We can use a third table to match every customer to every item. Note the tiny sample tables used here mean 6 x 7 = 42 records in this table. The real problem would generate about 140,000

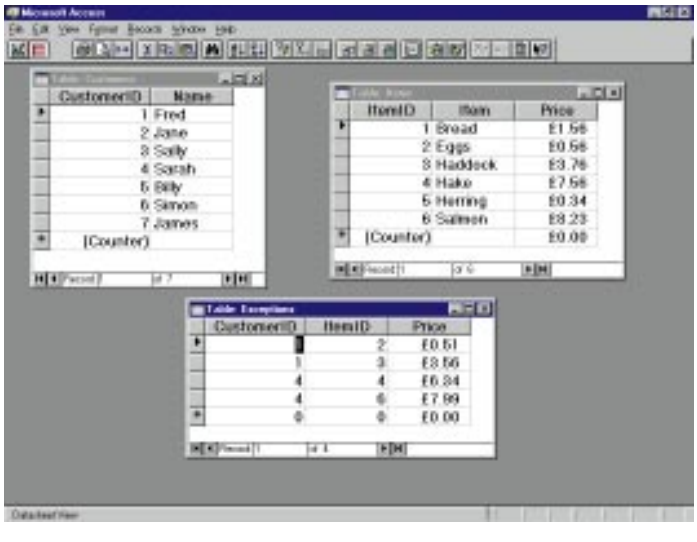


Fig 4 (right) Two tables store the basic data, and a third stores the exceptions to the rule

special price on, say, eggs, while another might have a special price on milk.

Given the logic of database design, my solution was to try to have three or four price lists calculated as queries and to have a field in the customer table assigning each client to a particular price level. However, this does not provide the client with the flexibility he requires, nor is he satisfied with the idea of combining the last solution with, say, a special overall discount (this being stored in the customer table).

Given that he has some 400 customers and 300+ products, the whole thing is becoming a major headache. Any suggestions, apart from aspirins or suicide, would be immensely welcome."

Mark Squire

Hmm… hopefully no aspirin or suicide required. This is an excellent problem because it is one example of a generic class of problem and as such is worth examining in some detail. As usual, the solution is shown in Access but could be implemented in any RDBMS.

Let's assume that we start with two entities: Customers and Items. Each gets their own table, as shown in Fig 2. Just for now, let's assume that all customers pay the same price for each item even though we know it isn't true. So in this case, the price would be an attribute of the entity Item and would therefore be placed in the same table, as shown.

Now let's assume that each customer negotiates a unique price for every item and that there is no standardisation whatsoever (equally untrue). In this case, we would typically generate a third table which tied the first two together and we would move the prices into that table (Fig 3). In practice, given 400 customers and, say, 350 products, this will be 400 x 350 = 140,000 records in the joining table. Big, but necessary.

These two approaches represent opposite ends of the spectrum. At one end, each item has but a single price. At the other end, each customer-item interaction has a price, and we position the pricing information accordingly. Mark's problem is that the real-world problem he is trying to model falls somewhere in-between the two. Most of his customer-item interactions use the default price, and a few are exceptions.

One answer is to put the default prices back into the Items table and create an Exceptions table which stores the exceptions to the defaults. Never let it be said that I can't pick suitable names for my tables.

This stores all the data in a reasonably efficient manner (Fig 4). As far as I can see, remembering from a couple of issues ago that this is an art, not a science, there is not much duplicated data here. So that's the problem solved, isn't it? Well, "yes" in terms of storage, but a big "not yet" in terms of implementation. How do we actually look up the price of an item for a particular customer?

A reasonable question is, "How would we do it if this were a paper-based

p283 ➢

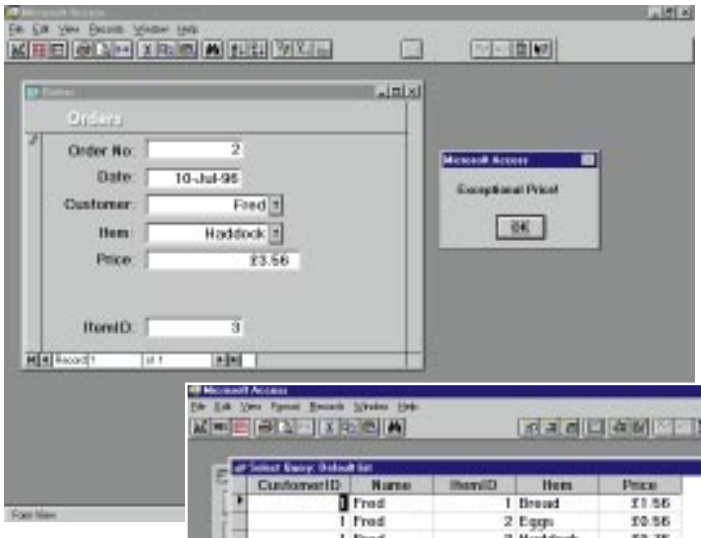


Fig 5 This form ties the Items and Exceptions tables together using code tied to the "After Update" Property of the Item combo box. The code pops up a message box telling you the price is an exception

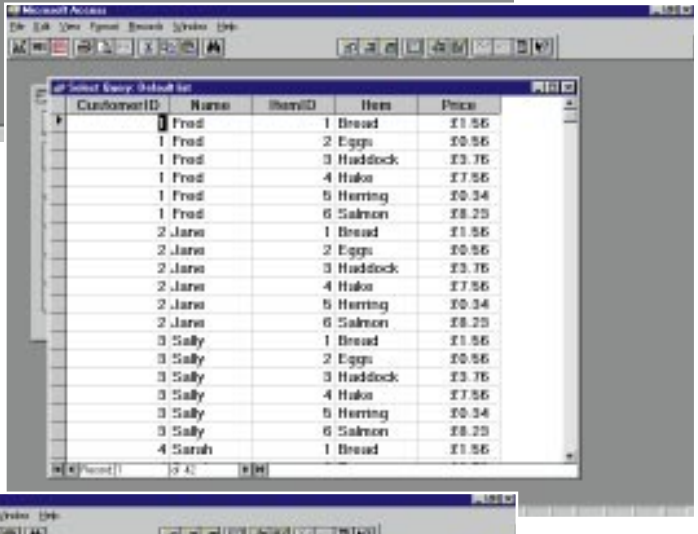


Fig 6 This query shows every default price for every customer. Note that there are seven customers and six items, so this table has 42 records

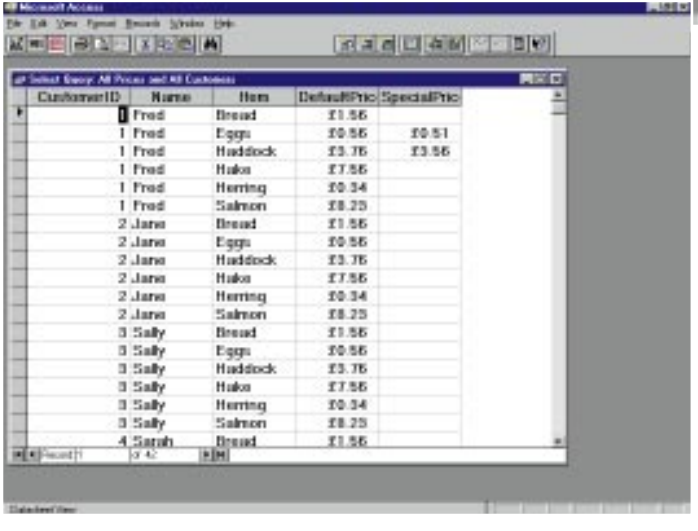


Fig 7 This query adds in the special prices where appropriate. Could be better though: any ideas?

Items and Exceptions

```
SELECT DISTINCTROW Customers.CustomerID, Customers.Name, Items.ItemID,
Items.Item, Items.Price
FROM Customers, Items
ORDER BY Customers.CustomerID, Items.ItemID;
        followed by
SELECT DISTINCTROW [Default list].CustomerID, [Default list].Name,
[Default list].Item, [Default list].Price AS DefaultPrice,
Exceptions.Price AS SpecialPrice
FROM Exceptions
RIGHT JOIN [Default list] ON (Exceptions.CustomerID = [Default
list].CustomerID) AND (Exceptions.ItemID = [Default list].ItemID);
```

Fig 8 Generating a table from Items and Exceptions

system?" If a customer ordered an item, we'd look first in the exceptions list to see if there was a special price. If not, we'd look in the standard price list and use the price shown there.

This effectively defines the algorithm I have used in the form shown in Fig 5. You select the Customer using the first combo box and then the Item with the second. A block of code runs whenever the second combo box is updated, which says:

1. Open the Exceptions table.
2. Search for an entry which has this customer and this item.
3. If an entry is found, copy the price from that record into the price field on this form.
4. If not, open up the Item table, find the correct item and copy the price from there.

This form is actually based on a simple Orders table, which records the date, customer ID, Item ID and Price. Please note that this is not a complete implementation since we all know that Orders are usually for more than one Item. The form is logically flawed at present. The price is only checked when the Item combo box is updated, so you can fool the system by selecting the Item and then changing the customer. However, it does demonstrate that the data can be pulled from the correct table in a manner which is transparent to the user.

Just out of interest, if we suppose that almost all of the prices were unique (which would then favour the use of a large joining table as described above), it might still be advantageous to keep the prices in the Items and Exceptions table. We could update the Exceptions table as changes occurred and then use a make-table query to generate the 140,000-record table which would be used on a day-to-day basis.

I was musing about the best way of generating such a table from Items and Exceptions, and the best I could come up with is shown in Fig 8.

The first query generates a list of all Customers and all default prices (Fig 6) while the second adds the special prices to that, where appropriate (Fig 7). I'd be the first to admit that it isn't perfect because it doesn't replace the default price with the special one when both occur in the same record. Can anyone come up with a better solution?

PCW Contacts

Mark Whitehorn welcomes readers' correspondence and ideas for the Databases column. He's on **m.whitehorn@dundee.ac.uk**

Fresh fields

Mark Whitehorn comments on areas of Microsoft's internet strategy relating to the future of DBMS. Plus, the TechNet troubleshooter CD-ROMs.

I have just returned from Microsoft's Tech Ed conference where Microsoft's internet strategy was much discussed. There are two areas of said strategy which are likely to be of major interest to DBMS fans.

The first is the rocketing growth in tools which provide web access to databases. Tools for Microsoft's Access and SQL server have already been released and the next six months will see rafts of the things appearing. What makes all this doubly exciting is that being a new field, there are no standards or yardsticks for comparison. So, for a while, we are all going to live in interesting times.

The second issue concerns the way in which all the data is controlled on the Internet. At Tech Ed I discussed this with Microsoft's Brad Silverberg (senior vice president, Internet Platform and Tools division), and he used Microsoft's TechNet (*see overleaf*) as an example. The data in TechNet isn't organised as a classical relational database; instead, it is mostly text and is organised more like a huge hypertext help system. This is currently distributed as a set of three CD-ROMs which are sent to each subscriber on a regular basis.

He said that this type and quantity of data should be on the Internet. We might envisage a situation where the data was originally distributed on CD-ROMs to each subscriber, transferred to writeable media and then subsequently updated from a web site. The problem, as Brad pointed out, is that this huge volume of data is not only expanded month by month, but additionally the existing data is edited and updated. Currently there are no standards on the internet for flagging changes to data with time, so working out a system for downloading just the changes is a nightmare.

Well, there may be no standards on the Internet, but this problem has been well understood in the database world for years. In fact, it is a small and relatively simple subset of the problems which can occur when data is replicated across two or more sites. All of which suggests to me that we can expect to see two distinct changes over the next few years.

Currently, web sites store essentially static data. As tools become available to tie RDBMSs to web servers, we will rapidly move to a situation where web sites display mainly static data with some embedded dynamic data which will come from an RDBMS. Some sites already do this. However, I don't think we'll stop there.

The web is going to become the repository for mind-bogglingly huge amounts of data. Unless that data is properly organised, it will rapidly become unmanageable. I believe that we will shortly see tools arriving which combine RDBMS and web server functionality. This combination will be conceptually very different from linking because the RDBMS will take over the management of *all* the data that the server presents to the world. With a tool like that, management of the information in TechNet will be simple.

The good news is that it looks like our databasing skills will be needed for many years yet, solving all of these new database problems.

In-flight databasing

I was both interested and exasperated to find myself beset by another database problem on my return journey. It was the old story: breakfast in Nice, lunch in Edinburgh, luggage in limbo. I didn't see it again until the following day. The enforced separation from my dirty socks for 24 hours didn't worry me; what concerned me was the cavalier manner in which airlines use, or do not use, the information they hold.

Airlines manage massive amounts of information about discrete items of data: passenger details, seat allocations and luggage details. For years they have managed to control the passenger and seat allocations reasonably well (failure to do so creates fist fights in the aisles, which are bad for PR). The control of luggage was traditionally less well implemented until the arrival of terrorist activities. It is now, shall we say, "politically incorrect" for an airline to be uncertain of the whereabouts of a given piece of luggage.

Many of us have been held inside a stuffy aircraft while the airline tries to find that last elusive passenger and tells us that, unless he or she is found, the entire luggage hold will have to be eviscerated. I have never complained, because it tells me that the airline knows which pieces of luggage are on the aeroplane and I'm duly grateful: by extrapolation we can assume that it also knows which luggage *isn't* on board.

So the database problem is easy. You know which passengers are in which seats, and which are unlucky enough to be parting from their luggage at 500 knots: it should only be a trivial task to send the aircraft, in flight, a list of those hapless passengers so that they can be informed before arrival.

What actually happens is that you, rather than the airline, do the database work. You hang around querying the carousel until it returns a null; in my case, this took about half an hour. Then you submit a query to the baggage reclaim server, only to discover that there is a FIFO (First In First Out) queue and you are L (Last). When your query reaches the head of the queue, you discover that the server performed a replication with London some

On my bookshelf

***The SQL Server Handbook — A Guide To Microsoft Database Computing**, by Ken England and Nigel Stanley.*
Covers SQL Server version 6.0, and although 6.5 has just arrived, a large proportion of the book is still relevant. Database servers are substantially different from PC RDBMSs, so for the many people who are currently moving up, such a book has to be well worth considering.

The authors avow that it is "definitely not intended to be a re-hash of the documentation set", which is certainly true; instead the book concentrates on the working principles behind SQL Server. Thus, instead of simply telling you *how* to create a device, it explains *what* a device is, and why you need them. The subject areas covered range from devices, through data integrity and database concurrency, to database administration and integration with Access and VB.

In general, the book is well laid out and readable. I had only two real criticisms. Firstly, it tends to favour the command line. For example, we are told that a device can be created using TRANSACT-SQL (after first using "sp_helpdevice" to identify an unused device number). The syntax is:

```
disk init
name = 'authors'
physname = 'c:\sql\data\aut.dat'
vdevno= 2
```

etc…

After all that, there is a brief note that using the GUI tool SQL Enterprise Manager will, among other virtues, locate an unused device number for you and specify it automatically. I don't know about you, but I'd choose a GUI tool any day.

Secondly, there isn't enough text devoted to the mechanics of driving a server. For example, in the above section, we are told that we "need to obtain a device number using the system procedure 'sp_helpdevice'". What the novice user *isn't* told is how or where to issue that command.

Nevertheless, this book offers an invaluable background. Considering the amount that you will be spending on the server, the expense must be worthwhile.

● *The SQL Server Handbook — A Guide To Microsoft Database Computing* by Ken England and Nigel Stanley. Digital Press, ISBN 1 - 781555 - 581527. Available from ICS Solutions (01256 469460). Price £29.99.

On my bookshelf

The online help in SQL Server is as good as online help ever seems to be, which is why a text-based book is still desirable

time ago and already knows about your luggage… ARGHHH! Let's use the technology properly, guys.

TechNet troubleshooters

For those who haven't come across it yet, Microsoft has compiled an excellent source of information onto a series of three CD-ROMs entitled TechNet. They're a mine of troubleshooting information about a range of Microsoft products. Be warned, though; it takes a pretty hefty outlay for access to this resource *(see PCW Contacts, page 273)*.

The extracts which follow give a flavour of the kind of information you'll find:

Q. *When I add two tables to my query that do not have a defined relationship, Microsoft Access automatically joins them. Can I prevent this from happening?*

A. Microsoft Access 2.0 automatically joins two tables in a query if the tables meet the following criteria:

- There is no relationship defined between the tables.
- Each table contains at least one field whose name and data type matches the name and data type of a field in the other table.
- One of the tables has a primary key defined on the matching field.

Only one AutoJoin is automatically created between two tables. Even if there is more than one join possible between the two tables, a join is created only between the first fields that meet the above criteria. If you add three tables that meet the above criteria to a query, three joins are created; one for each table pair. You cannot turn off this functionality: you must either delete the join line after it has been created, or manually define a relationship between the two tables.

Q. *Why can I update more fields in my query than I could in Microsoft Access version 1.x?*

A. In Microsoft Access 2.0, when a query includes fields from more than one related table, you can update data on both sides of the join. This means that in a query which combines data from two tables, you can update data from both of those tables.

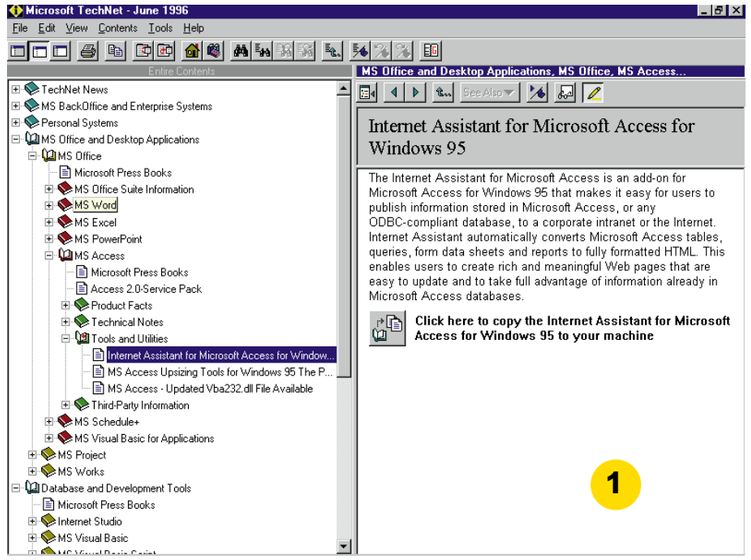


Fig 1
In full swing: you can use this to download new and exciting add-ons which are supplied on the TechNet CD

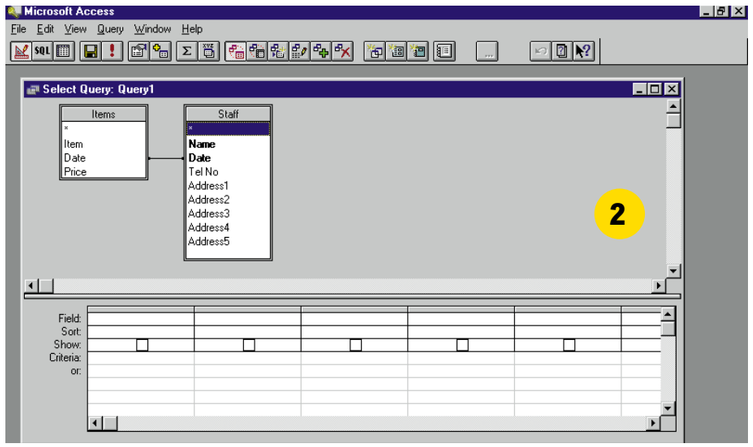


Fig 2
TechNet is also a source of riotously interesting information

Fig 3
The behaviour described in the first extract from TechNet — Microsoft Access, making joins between tables in a query. In some cases (such as this) the assumptions it makes may not be exactly what you had in mind

If you want to prevent users from updating fields in a multiple-table query, create a form based on the query and then set the Locked property for the fields you do not want users to update.

Q. *Why do I see a number instead of "(counter)" for my counter field?*

A. Microsoft Access 2.0 enters a counter value when you start to edit a new record. In Microsoft Access 1.x, this value was entered after you saved the record. Since this value is now provided earlier, if you start editing a new record and then cancel it, the counter value is still used even though no record is stored with the value. For example, when you add a new record to a table containing two records, the counter value is three. If you cancel this new record and later add another new record, the counter value is four for the new record.

Counter values are not re-used when you delete records. For example, if in a table of 15 records you delete the last three records and then add a new one, the counter value for this new record is 16. To reset the next available counter value, compact the database. After the database has been compacted, the next available counter is set to one higher than the last counter value in the table.

Q. *Why is the data sorted in my query but not in my report?*

A. Reports create their own internal queries to present the data. If you want to present the data in your report in a particular order, you must set the sort order explicitly in the Sorting And Grouping dialogue box. To do this, open the report in Design view and then choose Sorting And Grouping from the View menu.

Q. *How can I keep a group of records together in a report?*

A. The new KeepTogether property for groups in Microsoft Access version 2.0 gives you the ability to keep groups of like information together. This property is available in the Sorting And Grouping dialogue box for reports. Using this property, you can keep an entire group together (including the group header, all records and the group footer), or keep the group header with the first record.

Q. *Why is every other page of my report blank, and how can I correct this problem?*

A. The problem occurs when the total width of your report exceeds the width of the paper specified in the Print Setup dialogue box. For example, blank pages print if your report form is 8ins wide and your left and right margins are 1in wide for a total width of 10ins, and if the paper size specified in the Print Setup dialogue box is only 8.5ins wide.

Using this example, if controls (such as text boxes) extend beyond 8.5ins, the controls are printed on a second page. Otherwise, you receive a warning message stating that some pages may be blank. Blank pages generated after the warning are not counted in the total pages of your report.

Windows

The following information applies to Micro-soft Access versions 1.0, 1.1, 2.0 and 7.0:

Symptoms: *Using a make-table (SQL Select...Into) or an append (SQL Insert...Into) query with criteria that have no matching records, causes an empty table to be created. In Microsoft Access versions 1.x and 7.0, if the empty table contains a counter field, the first record added to the table will have a counter value of one. In Microsoft Access version 2.0, the first record added will have a counter value of two.*

Cause: Microsoft Access version 2.0 provides the first value for auto-increment fields internally. Because it provided 1 internally for the previous auto-increment column, the next value is 2, which shows in the new table.

Resolution: In version 2.0, create the table manually, instead of using a make-table query or append query, to start the counter at 1.

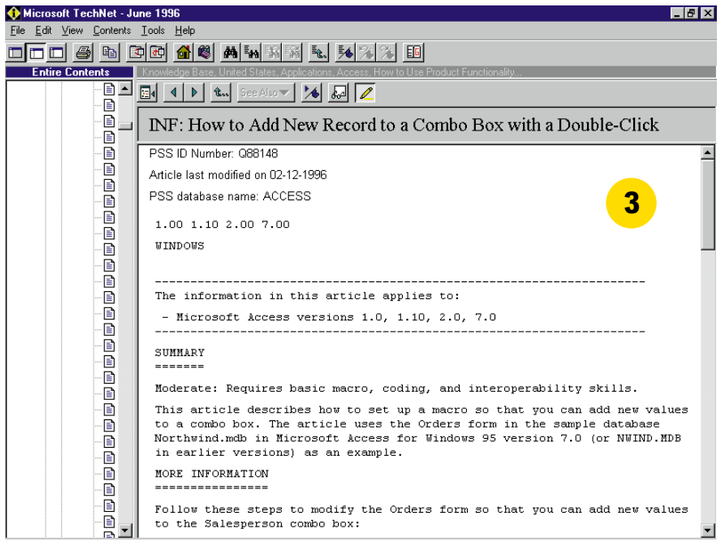
Status: This behaviour is by design.

Steps to Reproduce Behaviour:

1. Open the sample database Northwind.mdb (or NWIND.MDB versions 1.x and 2.0).
2. Create a new query based on the Employees table.
3. On the Query menu, click Make Table. Enter "Empty Table" (without the quotation marks) in the Table Name box, and then click OK.
4. Drag the EmployeeID and LastName fields to the QBE grid.

(Note: In versions 1.x and 2.0, there is a space in the field names Employee ID and Last Name.)

5. In the Criteria row for the EmployeeID column, enter "<1" (without the quotation marks).
6. Run the query. Note that a new, empty table called Empty Table is created.
7. Open the Empty Table and enter a name in the LastName field. Note that the counter starts at two instead of one in version 2.0. In versions 7.0 and 1.x, the counter starts at one, as expected.



PCW Contacts

Mark Whitehorn welcomes readers' correspondence and ideas for the Databases column. He's at **m.whitehorn@dundee.ac.uk**

MS TechNet costs £249 per year for a single user; £550 per year for the server-based version with unlimited access. (Both prices excl. VAT). **0800 281221**

Back to **normality**

The mighty meter problem is finished off once and for all as Mark Whitehorn covers normalisation of the original table and the fastest SQL solution.

Over the last two issues, we have looked at SQL (having started with the fundamental operators which underly the language). To my delight, the Editor has decided that SQL warrants wider coverage. So, next month, we will begin a separate three-part series on the subject, as a feature elsewhere in *PCW*.

This gives us more room in the column to look at other issues, which is handy because I want to finish off the continuing "meter problem" once and for all, (see the letter from Phil Bowles, below).

Meter for measure

I will briefly recap for the benefit of non-regular readers. In the March issue I posed a problem which my colleague, Stephen, had encountered in real life. It involved a table of readings from electricity meters; see the screenshot, *Table 1*. The primary key, in this case is made from [Meter No] and [Date].

The problem was that the people who produced the data also wanted to see it as shown in the screenshot, *Table 2*. This table shows data from two records in the same row and the apparently simple question was, how can you produce a table like that shown in the screenshot *Fig 1* from *Table 1* without offending the relational model?

Stephen and I found a workable way of deriving the second table from the first, but as our solution offended the relational model, I asked in this column if anyone knew of a solution which didn't cause such offense. I was grateful to be inundated by replies, several of which were published in the May issue.

However, two further points arose. One was normalisation. One respondent (as discussed in the June issue), suggested that the table I had originally used was not normalised and that this was part of the problem. He suggested the screenshot, *Table 2*, as an alternative. The primary key in this table is Reading No.

So, in the June issue I asked readers: which one do you think is flawed in terms of the relation model, and why?; to what update and delete anomalies does the flawed one lead?; which one will be faster when queried?; and what are the implications of using each table in a real database? The other issue was one of speed. Several people wanted to know; of the SQL solutions presented, which was the fastest?

In this issue we'll look at both of these areas, since both are relevant to more than merely the original question.

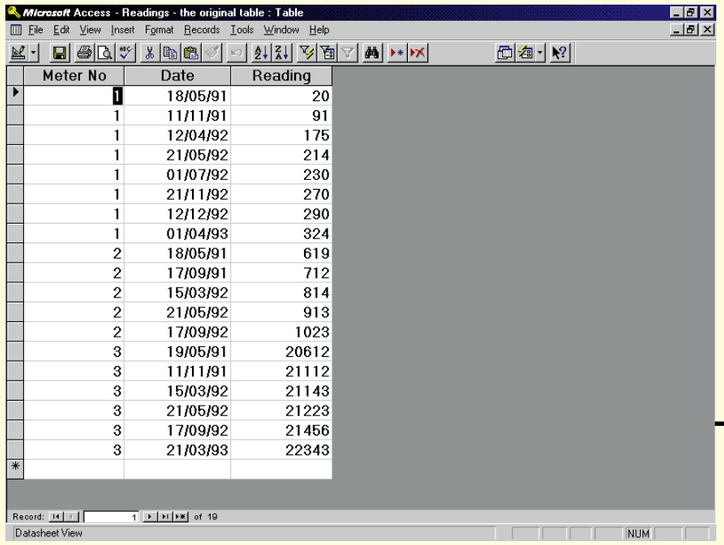


Table 1 *The primary key is compounded from Meter No. and Date*

Normalisation and data redundancy

By far the majority of readers who replied felt that Table 1 was properly normalised, as did Chris Date… yes, *the same*; the other half of "Codd and Date".

Regular readers will remember, that in the June issue a certain respondent suggested that I should "…go away and re-read Codd". Happily, I found I could do better than that because it was very

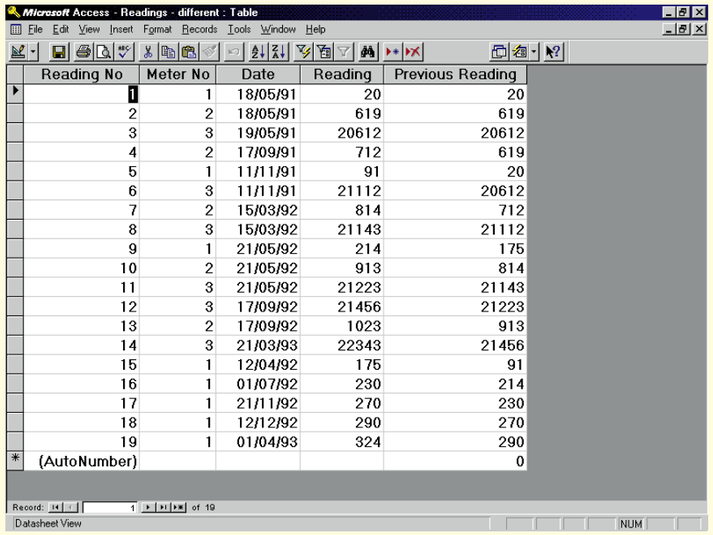


Table 2
The primary key is the "Reading No."

shortly afterwards that I met up with Chris Date. I couldn't resist showing him the tables and asking his opinion. In fact, we discussed three tables, the third being one where each record contains a pointer to the "previous" reading; as in the screenshot, *Table 3*.

Chris gave the following opinion: "We can ask ourselves 'what is the effect of normalisation?' Well, basically, it's to reduce redundancy but in order to consider that question carefully we have to have a careful definition of what redundancy is, and without getting into such a refined definition (because I don't think I could give you one), I will simply point out that normalisation *per se* does not, in general, eliminate all redundancy.

"Here's a classical example" (he indicated Table 2). "This is in third normal form and yet there is clear redundancy. What normalisation does (normalisation to the ultimate normal form) is to get you to a position that guarantees that you will not have any update anomalies that can be removed by taking projections. It doesn't say it'll get rid of all anomalies, it just gets rid of those anomalies which can be removed by taking projections.

So yes, you can have redundancy, and normalisation doesn't help. In fact, normalisation is the one tiny piece of science we have but it is not enough — there are all kinds of other questions — is this" (Table 2) "a good design or a bad design? — I don't know because it is subjective, there is no science there. The only sort of working definition of redundancy you can have is if, somehow, you can make something smaller: then you have redundancy. My gut feeling is that it's a bad design, but I can't quantify or qualify that really."

I think the information Chris gives here is well worth stressing, if only because several books that I have on database design get this wrong. Normalisation doesn't guarantee to remove all redundancy,it only removes that which can be removed by projection. Therefore, you can normalise a set of tables and still have redundancy and, hence, update anomalies lurking in the tables.

So, to answer my own questions: which is flawed in terms of the relation model? All three are in third normal form, but Table 2 contains redundant data, and both Tables 2 & 3 can suffer from update anomalies (see below). To what update and delete anomalies does the flawed one lead? Tables 2 & 3 have potential update and delete anomalies.

For example, consider Table 3. Suppose that we discover that meter no.1 was also read on 01/02/93 and yielded a reading of 300. We can add a record like this:

Reading no.*	Meter no.	Date	Reading	Prevs readg no.
15	1	12/04/92	175	5
16	1	01/07/92	230	9
17	1	21/11/92	270	16
18	1	12/12/92	290	17
19	1	01/04/93	324	18
20	1	01/02/93	300	18

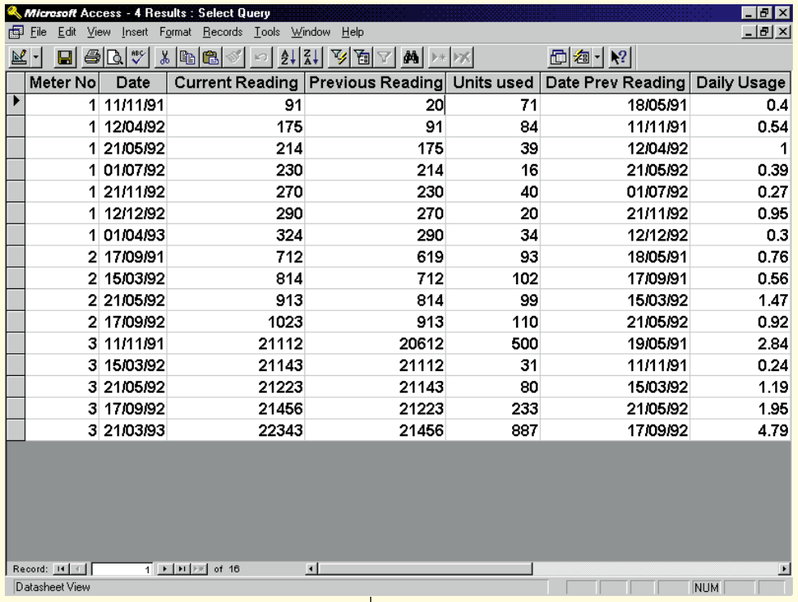


Fig 1 *The way in which the users want to see the data*

The fact that the row is "out of sequence" (at least, in terms of dates) is of no consequence. However, the addition of record 20 has rendered the pointer in record 19 incorrect (it now points to the wrong record). So unless we locate the errant record and correct it, the table now has an internal inconsistency.

In a nutshell this is what is wrong with this type of table design (in my view). Simple changes — updates, and by the same token, deletions — to, or of, one record can cause anomalies in other records. To ensure internal data integrity, some or all of the table has to be checked for integrity after every update. This is clearly not impossible to do but does makes extra work for the developer and may well slow down the database, particularly in a multi-user environment.

In addition, even if the developer's work is perfect, later maintenance work on the database may unknowingly circumvent the checks and lead to a loss of integrity.

Incidentally, the same generic problem exists with Table 2:

Reading No.*	Meter No.	Date	Reading	Prevs. Readg.
15	1	12/04/92	175	91
16	1	01/07/92	230	214
17	1	21/11/92	270	230
18	1	12/12/92	290	270
19	1	01/04/93	324	290
20	1	01/02/93	300	290

Which one will be faster when queried? It is clear that Table 2 will be the fastest for queries like the one desired by the people who wanted the original table. However, if the necessary checking is performed after updates, it will be slower to update.

What are the implications of using each table in a real database? To summarise, Table 1 makes the maintenance of data integrity much easier: queries run against Tables 2 and 3 should run queries more rapidly.

My feeling is that I would rarely implement a base table like 2 or 3, and most readers agreed, although many, like correspondent Phil Bowles, gave an impressively balanced view:

"It may well be that with all those things considered, his choice of solution is a shooting offence. Who can tell? Personally, I try to adhere to a clean design at the outset as experience shows me that it prevents major problems in the future and I'm clever enough to cope with complicated SQL — so I'd go with you. But then again, if I had a team of trainee programmers who were SQL-illiterate, I might consciously make the compromise, adulterate the design and simplify the SQL to reduce development problems.

Its all so complicated, isn't it? These are the reasons that I am no longer an IT professional — at the end of the day, who cares? This type of 'holy war' is one of the reasons why I left IT to become a police officer. If there's going to be a row, let it be over something that matters."

Be warned, he closed his email with *"So my final answer is: I don't care who is right or wrong but if you don't stop arguing right now, someone's going to get nicked."*

…OK, guv', fair enough.

The truth is, of course, that both of these table designs have advantages. The good news is that with a bit of extra work we should be able to have our speed *and* our data integrity.

Suppose we store the data in Table 1, and use that table for all data entry, updates and deletes. Suppose also that, every night, we run a background process that generates Table 2 from Table 1 and writes it to disk. We can then run the user queries against Table 2, and they will run like greased lightning.

Certainly there are disadvantages. The queries that we run under this regime have the potential to yield answers which are a maximum of one day out of date. So we might run the update of Table 2 twice a day, or three times a day — the important point is to discuss it with the users and discover what are acceptable limits. We can even offer them two alternatives: fast queries (run against Table 2) where the data might be slightly out of date; or slower ones (which run against Table 1) which are guaranteed to be up to date. In essence, what I am suggesting is a very simple form of data warehousing. It combines the best of both worlds, which is why it has become so popular recently. Trendy isn't necessarily bad.

Speed

It is worth stressing that in the original question all those months ago, I didn't ask for a *rapid* solution. I never mentioned speed at all, I asked for elegance, academic purity, relational correctness, whatever you want to call it, but not speed. So, while some of the original replies turn out to be slow, this in no way reflects badly upon either the "worth" of the original reply or the worth of the people who supplied them.

However, you asked for the relative speeds so here they are. I looked at three solutions. The first (labelled 4-Stage SQL) is the original one that Stephen and I concocted. It's crude, it's messy, it offends the relational model, but it works. The second is a two stage solution which used two SQL statements and the third is a single SQL solution. (Both of these solutions were featured in the May issue).

Times to complete the queries are given in seconds.

Number of Records	Four-stage SQL	Two-stage SQL	Single SQL
100	2	3	18
200	3	6	79
400	4	18	395
1000	5	117	*
2000	8	553	
20000	58		

*(After 2,700 seconds, the completion bar for the query was showing one percent. I know those bars are notoriously inaccurate but even so I thought the point had been made that it was very slow. I stopped the test because I needed the PC for something else).

Now see *Fig 2*; the obvious implication from these results is that the original, rather offensive, solution, happens to be very fast. The single stage SQL, while interesting, is very slow, even compared to the two-stage one. However there is a more fundamental difference. It is clear (if we invert the figures) that the efficiency of the "pure" solutions, in terms of the number of records processed per second, drops as the size of the table increases.

Number of records processed per second

Number of Records	Four-stage SQL	Two-stage SQL	Single SQL
100	50	33	6
200	67	33	3
400	100	22	1
1000	200	8	*
2000	250	4	
20000	345		

By contrast, the efficiency of the original method measured in these terms, actually increases. I must admit, that this answer surprised me initially, because I expected the set operations to be inherently faster. However, a little thought suggested an answer.

The original method still uses set operations — they are only "impure" in terms of the relational model. The great advantage for this method is that it simply has to manipulate tables of the same size as the test data. The problem for both of the "pure" solutions is that they involve self-joins. These in turn are generating huge intermediate tables which are presumably increasing in size by something like the square of the number of records. I suspect it is this that is gluing up the processing.

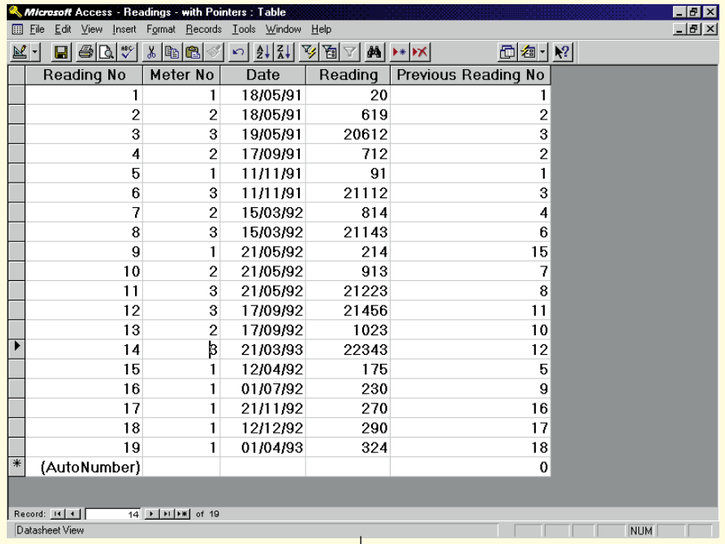


Table 3 *An alternative to Table 2, which essentially uses pointers rather than the data from the previous record*

Fig 2 *The results of the speed tests*

For those who still maintain an interest, and wish to speed test their own solutions, I have included an MDB file, on our cover-mounted CD-ROM, which is the testing database that I used. It is crude and essentially undocumented, because I developed it for my own use. Nevertheless, I suspect that readers who are competent enough to try their own SQL solutions will be able to drive it.

There is a form, with associated code, which will generate the test data for you. If you can find a really fast, yet still pure, solution let me know — but please, only if it is significantly faster!

● *For more about SQL, see the new feature series starting in next month's issue.*

PCW Contacts

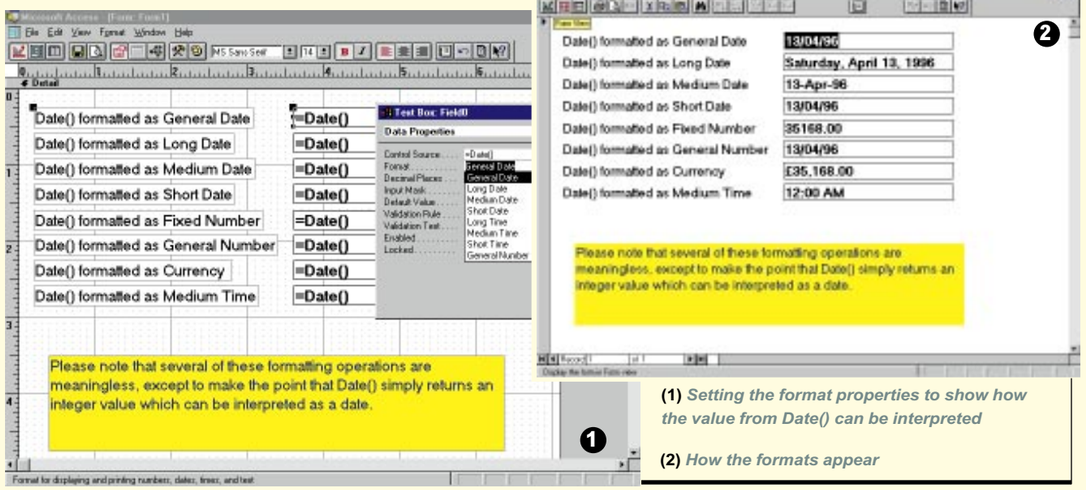
Mark Whitehorn welcomes readers' correspondence and ideas for the Databases column. He's on **m.whitehorn@dundee.ac.uk**

Do you have problems with **dates?**

Mark Whitehorn **can't help you with your love life, but date/time types are another matter.**

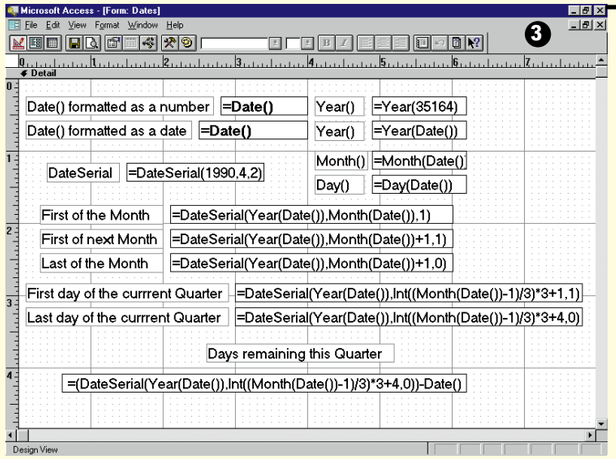
I have received several questions about handling dates in Access, particularly about ways in which specific dates (like the first day of the quarter) can be found. The following includes some elegant examples that I culled from the FAQs on the Microsoft Web site.

The Date/Time data type in Access is stored as a double-precision, floating-point number. The integer part represents the date and the decimal part represents the time. Clearly, we are only concerned with the integer part during this discussion.

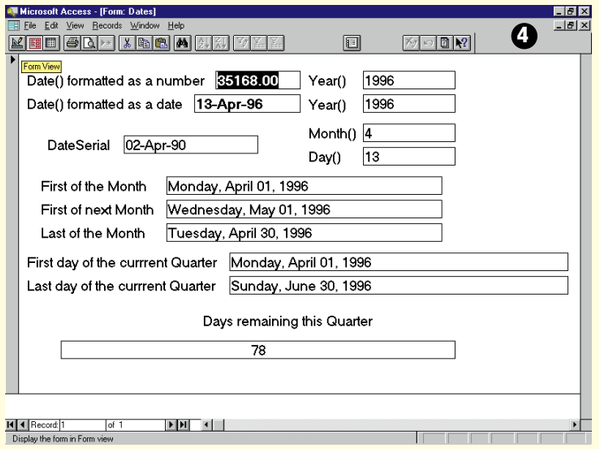


(1) *Setting the format properties to show how the value from Date() can be interpreted*

(2) *How the formats appear*



(3) *The full set of date manipulations described in the text...*



(4) *and how they appear, at least how they did appear on the 13th April*

Access includes several useful functions for date manipulation. For example, Date() returns the current date (as a number of course). This can be formatted to appear in a variety of ways on-screen (say, in a form) by choosing the appropriate format from the properties box *(see Figs 1 and 2)*.

Year() Month() and Day() are three functions which will extract the relevant part from any date. Without wishing to over-stress the point, this means that these functions will extract that information from a number since dates are stored as numbers. Clearly, these functions can be given an actual number (such as 31234), or they can be given Date() which in turn will provide them with "today's" number.

DateSerial() can be used to manipulate the day, month, and year components of a date. It takes three arguments and returns a serial version of the date. Thus:

```
DateSerial(1990,4,2)
```

returns

```
02/04/90
```

This can be presented on screen in different ways by playing with the format.

I realise that so far this list of functions and their abilities must sound a little tedious, not to say boring. However, given a working knowledge of these five functions, you can combine them in such ways as heaven's wonders to perform.

For example, to find the first day of the current month, you can use:

```
=DateSerial(Year(Date()),
 Month(Date()), 1)
```

The first of the next month:

```
=DateSerial(Year(Date()),
 Month(Date()) + 1, 1)
```

The last day of the current month (a clever one this!):

Fig 5

EMPLOYEES

Employee No	First Name	Last Name	Date Of Birth	Date Employed
1	Bilda	Groves	12/04/56	1/5/89
2	John	Greeves	21/03/67	1/1/90
3	Sally	Smith	1/5/67	1/4/92

Fig 6

SALES

Sale No	Employee No	Customer	Item	Supplier	Amount
1	1	Simpson	Sofa	Harison	£ 235.67
2	1	Johnson	Chair	Harrison	£ 453.78
3	2	Smith	Stool	Ford	£ 82.78
4	2	Jones	Suite	Harisonn	£3421.00
5	3	Smith	Sofa	Harrison	£ 235.67
6	1	Simpson	Sofa	Harrison	£ 235.67
7	1	Jones	Bed	Ford	£ 453.00

Fig 7

SALES2

Sale No	Employee No	Customer	Item	Supplier	Amount
3	2	Smith	Stool	Ford	£ 82.78
5	3	Smith	Sofa	Harrison	£ 235.67
213	3	Williams	Suite	Harisonn	£3421.00
216	2	McGreggor	Bed	Ford	£ 453.00
217	1	Williams	Sofa	Harrison	£ 235.67
218	3	Aitken	Sofa	Harison	£ 235.67
225	2	Aitken	Chair	Harrison	£ 453.78

```
=DateSerial(Year(Date()),
 Month(Date()) + 1, 0)
```

The first day of the current quarter:

```
=DateSerial(Year(Date()),
 Int((Month(Date()) - 1) / 3) *
 3 + 1, 1)
```

The last day of the current quarter:

```
=DateSerial(Year(Date()),
 Int((Month(Date()) - 1) / 3) *
 3 + 4, 0)
```

Number of days remaining in this quarter:

```
=(DateSerial(Year(Date()),Int((Month
Date())-1)/3)*3+4,0))-Date()
```

and so on *(see Figs 3 and 4, page 293)*.

The possibilities are almost endless…

Gang screens

I don't want anyone to think I am obsessed with gang screens, but…

Windows 95

Right click on the DESKTOP, select NEW, FOLDER and name it "and now, the moment you've all been waiting for". Press Enter, right click on the folder and rename it to "we proudly present for your viewing pleasure". Press Enter again, right click and rename the folder once more to "The Microsoft Windows 95 Product Team!", now open the folder. (Just type in the words, not the inverted commas.)

Excel 95

First open a new Excel workbook and go to row 95. Select the entire row, then press tab once to move the cursor into column B (the entire row should remain selected). Pop down Help, About Excel, hold down CTRL and SHIFT together and select the Technical Support button. A new window will open. Walk forwards slightly, turn around 180 degrees, walk up to the wall and type "excelkfa". A secret door will open and I leave it up to you to navigate across the top of the wall to the next room.

What has this to do with databases? Er, the gang screens present data about the people who wrote the products, so the gang screens themselves must be databases. Sounds reasonable to me. *(But not to me — Ed.)*

SQL tutorial

Last month I started looking at SQL and began with the operators that it uses. We covered Restrict (aka Select), Union, Difference and Intersection. That only leaves two major ones, Product and Projection.

Once more, the sample tables are presented here *(Figs 5, 6 & 7)*.

Product

The product of two tables is a third which contains all the records in the first one, added to each of the records in the second. Thus, if the first table has 3 records and the second has 7, the product will have 21 records. The product of EMPLOYEES times SALES is shown in *Fig 8*.

This product operation has been applied quite correctly; however, the astute reader will note that the table in *Fig 8* contains seven rows which appear to be "meaningful" and 14 which are not. Note that we are dealing with a raw operator which takes no account of the values in fields, nor of any meaning that those values may imply or indicate.

In practice, the product operation may need to be modified by further operations in order to yield a meaningful answer.

The even more astute reader will have noticed that *Fig 8* contains two fields with identical field names. This state of affairs is not permitted in a table, and in practice an RDBMS will have to cope with this in some way, perhaps by renaming one of the fields.

However, as has been said before, these relational operators are the "primitives" from which more complex systems are constructed and it's usually the job of these higher-level constructs to cope with problems like this.

Projection

Projection selects one or more fields from a table and generates a new table which contains all the records, but only the selected fields. Thus, if we project EMPLOYEES on [FirstName] and [LastName] the result is as *Fig 9*.

This seems straightforward, but if we project SALES on [EmployeeNo] and [Customer] the result is as *Fig 10*.

Despite the fact that [SALES] has seven records, the answer table has only six. This is because one of them:

```
1          Simpson
```

would be duplicated in the answer table, and tables cannot contain duplicated records.

If we projected SALES on [SaleNo], [EmployeeNo] and [Customer] then the answer table *(Fig 11)* will contain seven records because in the original table the values in [SalesNo] are unique.

Fig 8

Employee No	First Name	Last Name	Date Of Birth	Date Employed	Sale No	Employee No	Customer	Item	Supplier	Amount
1	Bilda	Groves	12/04/56	1/5/89	1	1	Simpson	Sofa	Harison	£ 235.67
1	Bilda	Groves	12/04/56	1/5/89	2	1	Johnson	Chair	Harrison	£ 453.78
1	Bilda	Groves	12/04/56	1/5/89	3	2	Smith	Stool	Ford	£ 82.78
1	Bilda	Groves	12/04/56	1/5/89	4	2	Jones	Suite	Harisonn	£3421.00
1	Bilda	Groves	12/04/56	1/5/89	5	3	Smith	Sofa	Harrison	£235.67
1	Bilda	Groves	12/04/56	1/5/89	6	1	Simpson	Sofa	Harrison	£ 235.67
1	Bilda	Groves	12/04/56	1/5/89	7	1	Jones	Bed	Ford	£ 453.00
2	John	Greeves	21/03/67	1/1/90	1	1	Simpson	Sofa	Harison	£ 235.67
2	John	Greeves	21/03/67	1/1/90	2	1	Johnson	Chair	Harrison	£ 453.78
2	John	Greeves	21/03/67	1/1/90	3	2	Smith	Stool	Ford	£ 82.78
2	John	Greeves	21/03/67	1/1/90	4	2	Jones	Suite	Harisonn	£3421.00
2	John	Greeves	21/03/67	1/1/90	5	3	Smith	Sofa	Harrison	£235.67
2	John	Greeves	21/03/67	1/1/90	6	1	Simpson	Sofa	Harrison	£ 235.67
2	John	Greeves	21/03/67	1/1/90	7	1	Jones	Bed	Ford	£ 453.00
3	Sally	Smith	1/5/67	1/4/92	1	1	Simpson	Sofa	Harison	£ 235.67
3	Sally	Smith	1/5/67	1/4/92	2	1	Johnson	Chair	Harrison	£ 453.78
3	Sally	Smith	1/5/67	1/4/92	3	2	Smith	Stool	Ford	£ 82.78
3	Sally	Smith	1/5/67	1/4/92	4	2	Jones	Suite	Harisonn	£3421.00
3	Sally	Smith	1/5/67	1/4/92	5	3	Smith	Sofa	Harrison	£235.67
3	Sally	Smith	1/5/67	1/4/92	6	1	Simpson	Sofa	Harrison	£ 235.67
3	Sally	Smith	1/5/67	1/4/92	7	1	Jones	Bed	Ford	£ 453.00

Fig 9

ANSWER

FirstName	LastName
Bilda	Groves
John	Greeves
Sally	Smith

Fig 10

SALES

Employee No	Customer
1	Johnson
1	Jones
1	Simpson
2	Jones
2	Smith
3	Smith

Fig 11

ANSWER

SaleNo	Employee No	Customer
1	1	Simpson
2	1	Johnson
3	2	Smith
4	2	Jones
5	3	Smith
6	1	Simpson
7	1	Jones

Summary

The following is not rigorous, nor is it detailed, but if you have read and understood the previous section it should provide a quick reference to remind you what the operators are and what they do.

Two of the operators (**Restriction** and **Projection**) operate on single tables.

- **Restriction** (aka Select) extracts records.
- **Projection** extracts fields.

The remaining four operators (**Union, Difference, Intersection** and **Product**) all perform operations on two tables.

- **Union** adds the records from two tables together.
- **Difference** subtracts the records in one table from those in another.
- **Intersection** locates the records that are common to two tables.
- **Product** multiplies the records in the two tables together.

Assuming that each operation is performed on a pair of tables with 20 and 10 records respectively, the number of records in the answer table will have:

- **Union** — between 20 and 30
- **Difference** — between 20 and 10 (assuming that we subtract the table with 10 records from that with 20)
- **Intersection** — between 0 and 10
- **Product** — 200

Join

Join is often used as a relational operator and it can be built up from the simpler ones described earlier. Think of it as a mixture of the product and restriction operators, sometimes with an added dash of projection.

Suppose that you want to examine the sales that have been made by your employees. In order to do this, you need information from both the EMPLOYEES and SALES tables. (In fact, the table SALES2 contains information about more sales, and if we wanted to include this information we would first use the union operator. However, for the sake of brevity, we will assume that we are only interested in the sales recorded in SALES.)

Fig 12

Employee No	First Name	Last Name	Date Of Birth	Date Employed	Sale No	Employee No	Customer	Item	Supplier	Amount
1	Bilda	Groves	12/04/56	1/5/89	1	1	Simpson	Sofa	Harison	£ 235.67
1	Bilda	Groves	12/04/56	1/5/89	2	1	Johnson	Chair	Harrison	£ 453.78
1	Bilda	Groves	12/04/56	1/5/89	6	1	Simpson	Sofa	Harrison	£ 235.67
1	Bilda	Groves	12/04/56	1/5/89	7	1	Jones	Bed	Ford	£ 453.00
2	John	Greeves	21/03/67	1/1/90	3	2	Smith	Stool	Ford	£ 82.78
2	John	Greeves	21/03/67	1/1/90	4	2	Jones	Suite	Harisonn	£3421.00
3	Sally	Smith	1/5/67	1/4/92	5	3	Smith	Sofa	Harrison	£235.67

Fig 13

Employee No	First Name	Last Name	Date Of Birth	Date Employed	Sale No	Customer	Item	Supplier	Amount
1	Bilda	Groves	12/04/56	1/5/89	1	Simpson	Sofa	Harison	£ 235.67
1	Bilda	Groves	12/04/56	1/5/89	2	Johnson	Chair	Harrison	£ 453.78
1	Bilda	Groves	12/04/56	1/5/89	6	Simpson	Sofa	Harrison	£ 235.67
1	Bilda	Groves	12/04/56	1/5/89	7	Jones	Bed	Ford	£ 453.00
2	John	Greeves	21/03/67	1/1/90	3	Smith	Stool	Ford	£ 82.78
2	John	Greeves	21/03/67	1/1/90	4	Jones	Suite	Harisonn	£3421.00
3	Sally	Smith	1/5/67	1/4/92	5	Smith	Sofa	Harrison	£235.67

Go-slow on speed

I have said that I'd look at the speed of the different SQL solutions to the long-running meter problem. However, last month I published another solution and asked for comments on both its match to the relational model and its speed potential. There is a delay between my writing this column and you reading it, such that as I write this month's you still haven't read last month's. So, I'll delay the speed issue one more month, and then let you know what I found.

The first job is to perform a projection on these tables. Next we need to perform a selection which removes the records where EMPLOYEES.EmployeeNo is not equal to SALES.EmployeeNo. (Finally, we might optionally remove some fields from the answer table.)

We might express a join in this form:

```
EMPLOYEES JOIN (EMPLOYEES.EmployeeNo
        = SALES.EmployeeNo) SALES
```

and the result would be as shown in *Fig 12*.

To be a little more accurate, the table in *Fig 12* is the result of what is known as an **equijoin**. The table in *Fig 13* is the result of a **natural** join.

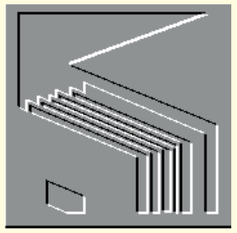
The simplistic difference is that one of the fields used in the join has been removed from the answer table.

There is slightly more to this than meets the eye, however. Joins come in several flavours and you will hear people talking about natural, equi, theta, outer and semi-joins.

While it is true that all of these joins differ in usefulness, they nevertheless all find their way into discussions about SQL. So, we'll have a look at them in more detail next month. ■

PCW Contacts

Mark Whitehorn welcomes readers' correspondence and ideas for the Databases column. He's on **m.whitehorn@dundee.ac.uk**



Meter **mania**

The meter readings problem has sent Mark Whitehorn's postbag level off the dial and offered up an alternative view.

The meter reading problem which first appeared in the March issue, continues to fascinate people (myself included). One reader is, however, convinced that I have missed the point. He feels that the real problem lies not in the SQL required to extract the data, but way back in the structuring of the original table I supplied. He is convinced that it isn't normalised to third normal form, and that the entire problem is easier to solve if the table is normalised to this level.

The original table structure I suggested for the meter problem…

Meter No	Date	Reading
1	18/05/91	20
1	11/11/91	81
1	12/04/92	175
1	21/05/92	214
1	01/07/92	280
1	21/11/92	270
1	12/12/92	290
1	01/04/93	824
2	18/05/91	619
2	17/09/91	712
2	15/03/92	814
2	21/05/92	918
2	17/09/92	1028
3	18/05/91	20612
3	11/11/91	21112
3	15/03/92	21143
3	21/05/92	21228
3	17/09/92	21456
3	21/03/93	22343

Challenging meter reader

The following is paraphrased from several email exchanges on the subject.

"The main problem stems from the fact that you have failed to identify that each meter reading is in itself an item of interest and should therefore have an individual unique identifier. It is often true that combining data fields in order to arrive at a unique index means that the data design is potentially incomplete. It can be useful to retrieve items in this way as it may speed up searching, but it is often more desirable to have a separate unique item key.

Reading No	Meter	Date	Reading	Previous Reading
1	1	18/05/91	20	20
2	2	18/05/91	619	619
3	3	18/05/91	20612	20612
4	2	17/09/91	712	619
5	1	11/11/91	81	20
6	3	11/11/91	21112	20612
7	2	15/03/92	814	712
8	3	15/03/92	21143	21112
9	1	21/05/92	214	175
10	2	21/05/92	918	814
11	3	21/05/92	21228	21143
12	3	17/09/92	21456	21228
13	2	17/09/92	1028	913
14	3	21/03/93	22343	21456
15	1	12/04/92	175	91
16	1	01/07/92	230	214
17	1	21/11/92	270	230
18	1	12/12/92	290	270
19	1	01/04/93	324	290

…and the table suggested by a reader. Which one is normalised? Which one will be faster to query?

"Thus, by adding (as a counter type) a ReadingID field at the front of your table and then the rather obvious PreviousReading at the end, we end up with a five-field table to deal with using ReadingID as the primary key.

"PreviousReading contains the ID of the previous reading and, to make the initial conditions easier, can point to itself if there are no previous readings."

At this point, the discussion moved to one about third normal form:

"Nowhere does it state (in Codd's rules) that you can simply add together other fields to make up your unique identifier. The use of a single unique key is indeed needed in order to satisfy third normal form.

"This sort of 'I can add things together to make it fit the rules' approach causes more problems that any other aspect of database design. Please go away and re-read Codd. You will see that the only way the data can be produced in third normal form is by the addition of the field I suggest, not by combining two other fields to make it work.

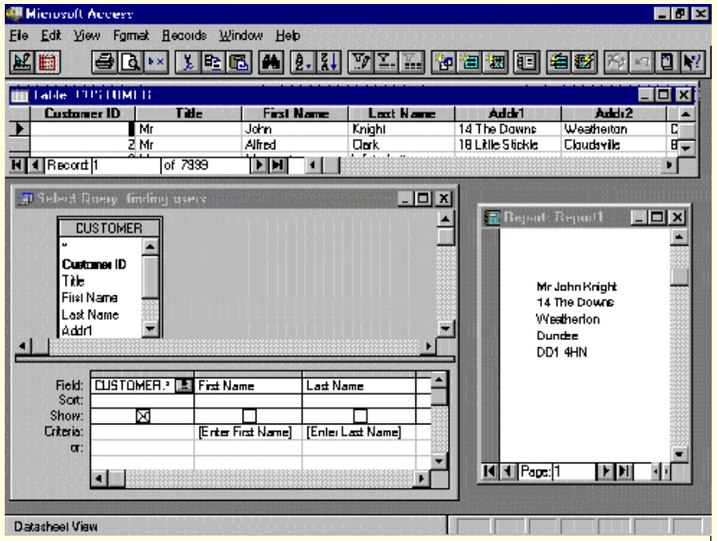
"The solution I proposed in fact meets third normal form, in the same way as yours does. You may just argue that this extra field is redundant as it reproduces the information you already have, but that is to miss the basic advantages and simplification that it gives. You cannot claim that it breaks third normal form. (If you consider that each unique ID only relates to one combination of Meter/Date and vice-versa, you will see this.)

"I repeat. If you have to add together two fields to make up a (primary) key, it is almost certain that you have missed the need for a unique identifier. Such was your case. Each record as taken by the meter-reading man stands alone as a separate item and should be identified as such. It may be convenient to refer to it by other fields, but that is not the point."

It is clear that we disagree fundamentally about the way in which this table should be structured in order to comply with the relational model. Inevitably, I think that I am right, but that proves nothing in absolute terms — have you ever met any database people who *didn't* think they were in the right?

More interesting than who is right or wrong is the fact that the two solutions provide an interesting compare and contrast exercise:

1. Which one do you think is flawed in terms of the relation model, and why?
2. To what update and delete anomalies does the flawed one lead?
3. Which one will be faster when queried?
4. What are the implications of using each table in a real database?



A parameter is easy to set up. Enclosing a criteria entry in [square brackets] means that, when the query runs, it will ask you for the relevant data (shown on the right of the screen)

Singled out

"Can you tell me how to get access to print one label instead of a complete list?"
ENSC1EM1@tay.ac.uk

The trick is to base your report on a query which lists only the single label that you want to print. In fact, if you want to be flash, you can base the report on a parameter query. A parameter query is one which asks you for the condition (for instance, the name of the person for whom the label is destined) before it runs.

So, when you run the report, it will run the parameter query; the parameter query will ask you for the name, and the report will print the label!

Sounds complicated, but it should be simple to use.

Table 1

EmployeeNo	FirstName	LastName	DateofBirth	DateEmployed
1	Bilda	Groves	12/04/56	1/5/89
2	John	Greeves	21/03/67	1/1/90
3	Sally	Smith	1/05/67	1/4/92

Table 2

SaleNo	EmployeeNo	Customer	Item	Supplier	Amount
1	1	Simpson	Sofa	Harison	£ 235.67
2	1	Johnson	Chair	Harrison	£ 453.78
3	2	Smith	Stool	Ford	£ 82.78
4	2	Jones	Suite	Harisonn	£3421.00
5	3	Smith	Sofa	Harrison	£ 235.67
6	1	Simpson	Sofa	Harrison	£ 235.67
7	1	Jones	Bed	Ford	£ 453.00

Table 3

SaleNo	EmployeeNo	Customer	Item	Supplier	Amount
3	2	Smith	Stool	Ford	£ 82.78
5	3	Smith	Sofa	Harrison	£ 235.67
213	3	Williams	Suite	Harisonn	£3421.00
216	2	McGreggor	Bed	Ford	£ 453.00
217	1	Williams	Sofa	Harrison	£ 235.67
218	3	Aitken	Sofa	Harison	£ 235.67
225	2	Aitken	Chair	Harrison	£ 453.78

Table 4

SaleNo	EmployeeNo	Customer	Item	Supplier	Amount
1	1	Simpson	Sofa	Harison	£ 235.67
6	1	Simpson	Sofa	Harrison	£ 235.67

● *Please email me, or write in and let me know what* you *think.*

Is it a bird? Is it a bug?

The question of speed in the meter problem is also of some interest. The different solutions suggested clearly have different speed implications and different update potentials. I started (as promised) to have a look at this, but I have been unable to come to a final conclusion for the intriguing reason that some of the tests are hanging Access. Is it a bird, is it a bug, is it my PC? Microsoft and I are, as we say in the trade, engaging in meaningful discussions. I'll keep you posted.

And so, to SQL

It was clear from the meter problem that many people wanted the subject of SQL covered for its own sake, so here we go. But first, a little background.

Data manipulation is a really important part of the relational model. After all, there is little point in storing data correctly, safely and securely, if that is all you ever do with it. Stored data has no value if you cannot question it and extract the data in some way for humans to examine. SQL is the language typically used for data manipulation and this, like most languages, has a set of commands. These commands in their turn are based on a set of fundamental operators.

In order to get the best from SQL, it is worth having at least a passing acquaintance with these operators. They are rarely used directly, but in my experience they keep cropping up in conversations about SQL. Knowing nothing about them can leave you at a social disadvantage. Trust me, it's worth knowing.

The only qualifier I need to apply is that the definitions I give are not complete to the relational model. For example, I say below that: "In order for tables to be 'Union Compatible', they must have the same number of fields, and the fields must be of the same data type and size." In fact, to be truly Union compatible, the fields must also draw their values from the same domains.

However, this level of pedantry is counter-productive, since most RDBMSs do not even support domains. So, as in the past, I will sacrifice exact mapping on to the relational model for practicality/readability.

The relational operators

Most of us are familiar with the standard algebraic operators (+,-,* and /) which signify addition, subtraction, multiplication and division. We use these operators to manipulate numerical values, or variables which represent values, almost without thinking. Thus, if we know that

```
A=5, B=6 and C=10
```

and that

```
D=A+(B*(C/A))
```

we can calculate that

```
D = 5+(6*(10/5))= 5+(6*2) = 5+12 = 17
```

In a database, we store the data in

If I enter "John" in answer to the first question, and "Knight" in answer to the second, then the parameter query will find his one record out of 7999. The report is based on the query, so it prints out only a single address

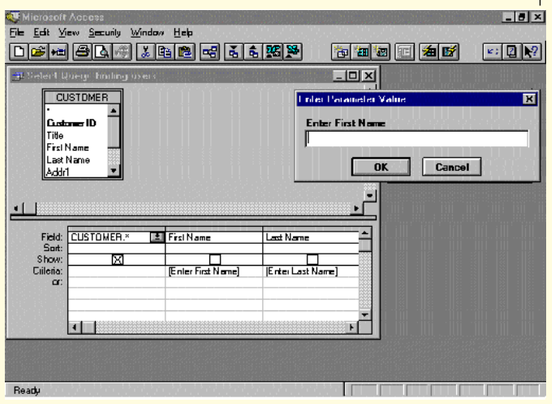


Table 5

SaleNo	EmployeeNo	Customer	Item	Supplier	Amount
1	1	Simpson	Sofa	Harison	£ 235.67
2	1	Johnson	Chair	Harrison	£ 453.78
3	2	Smith	Stool	Ford	£ 82.78
4	2	Jones	Suite	Harisonn	£3421.00
5	3	Smith	Sofa	Harrison	£ 235.67
6	1	Simpson	Sofa	Harrison	£ 235.67
7	1	Jones	Bed	Ford	£ 453.00
213	5	Williams	Suite	Harisonn	£3421.00
216	2	McGreggor	Bed	Ford	£ 453.00
217	1	Williams	Sofa	Harrison	£ 235.67
218	4	Aitken	Sofa	Harison	£ 235.67
225	4	Aitken	Chair	Harrison	£ 453.78

Table 6

SaleNo	EmployeeNo	Customer	Item	Supplier	Amount
3	2	Smith	Stool	Ford	£ 82.78
5	3	Smith	Sofa	Harrison	£235.67

Table 7

SaleNo	EmployeeNo	Customer	Item	Supplier	Amount
1	1	Simpson	Sofa	Harison	£ 235.67
2	1	Johnson	Chair	Harrison	£ 453.78
4	2	Jones	Suite	Harisonn	£3421.00
6	1	Simpson	Sofa	Harrison	£ 235.67
7	1	Jones	Bed	Ford	£ 453.00

Table 8

SaleNo	EmployeeNo	Customer	Item	Supplier	Amount
213	3	Williams	Suite	Harisonn	£3421.00
216	2	McGreggor	Bed	Ford	£ 453.00
217	1	Williams	Sofa	Harrison	£ 235.67
218	3	Aitken	Sofa	Harison	£ 235.67
225	2	Aitken	Chair	Harrison	£ 453.78

Table 9

SaleNo	EmployeeNo	Customer	Item	Supplier	Amount
3	2	Smith	Stool	Ford	£ 82.78
5	3	Smith	Sofa	Harrison	£235.67

Table 10

SaleNo	EmployeeNo	Customer	Item	Supplier	Amount
3	2	Smith	Stool	Ford	£ 82.78
5	3	Smith	Sofa	Harrison	£235.67

tables (also known as relations) and the relational model provides a set of operators (known, therefore, as relational operators) with which we can manipulate tables (that is to say, relations). The discerning, sensitive reader will have noticed that I am showing a slight tendency to slip into "database-speak" at this point. I do so only because, without using the terms, expressions such as "Relational Operators" do not make much sense.

However, it is worth noting before we begin that tables are relations, and relations are sets. Four of the relational operators (Union, Difference, Intersection and Product) allow us to perform traditional set operations.

In order to demonstrate these operators we need a sample table or three *(see Tables 1, 2 and 3*).

Restrict (aka Select)

Restrict simply extracts records from a table. Thus, if we perform a restriction on the table SALES where (Customer) = "Simpson", the result would be as shown in *Table 4*.

Union

Union creates a new table by adding the records of one table to those of another. Clearly, for this to work well it is essential that the tables have the same structure. The union of tables *Sales* and *Employees* is unimaginable because the two tables are clearly very different in structure. In order for tables to be "Union Compatible", they must have the same number of fields, and the fields must be of the same data type and size. The tables (*Sales* and *Sales2*) are "union compatible" and the result would be as shown in *Table 5*.

Note that the two records shown in *Table 6* were shared by the two tables, but have appeared only once each in the ANSWER (*Table 5)*. The order in which records appear in the result of a union is unimportant, but duplicate records are eliminated.

Difference

The difference of two tables is a third table which contains the records which appear in the first but *not* in the second. The tables concerned must be union compatible. Thus the difference of SALES (*Table 2*) and SALES2 (*Table3*) is shown in *Table 7*.

Note that, unlike Union, the order of the tables is vital. Thus the difference of SALES2 and SALES is not the same *(Table 8*). However, the records that are "missing" from the two ANSWER tables are the same *(Table 9*). That is, in both cases it is the records that are common to both of the tables involved in the difference operation.

Intersection

The intersection of two tables is a third table which contains the records common to both of them. Thus, the intersection of SALES and SALES2 is shown in *Table 10*.

Unlike the difference operation, the order of the tables is unimportant, but, of course, the two tables must be union compatible. ▣

PCW Contacts

Mark Whitehorn welcomes readers' correspondence and ideas for the Databases column. He's available on **m.whitehorn@dundee.ac.uk**