Food for thought

Mark Whitehorn considers an alternative view to sort the wheat from the chaff in his pantry.

In the March issue I published a way of finding all the recipes that would match the contents of a larder. This sort of solution is, of course, applicable to more than food. You could, for example, use it to choose machines that can be built from a given set of components.

A different solution was sent in by Alan Mackechnie (mkechnie@ifb.co.uk). This works not by finding those recipes which have all the ingredients in the larder, but by finding all those recipes which don't have all the ingredients in the larder. This isn't as perverse as it first sounds because, once you have found the set of recipes that you don't want, all you have to do is to subtract them from the entire set and suddenly you have the ones you want.

His solution was written in generic SQL, which I've modified slightly to run under Access and I've also added a key to the larder table. The SQL reads as:

```
SELECT d1.dish
FROM dishes AS d1
WHERE d1.dishid not in
(select di1.dishid from ↙
[dish/ingredient] as di1
where di1.ingredientID not ↙
in (select ingredientID ↙
from larder));
```

(*Key: ↙ code string continues*)

Very neat, so how does it work exactly? Well, this particular SQL statement can be dismantled very neatly from the bottom up:

```
select ingredientID from ↙
larder
```

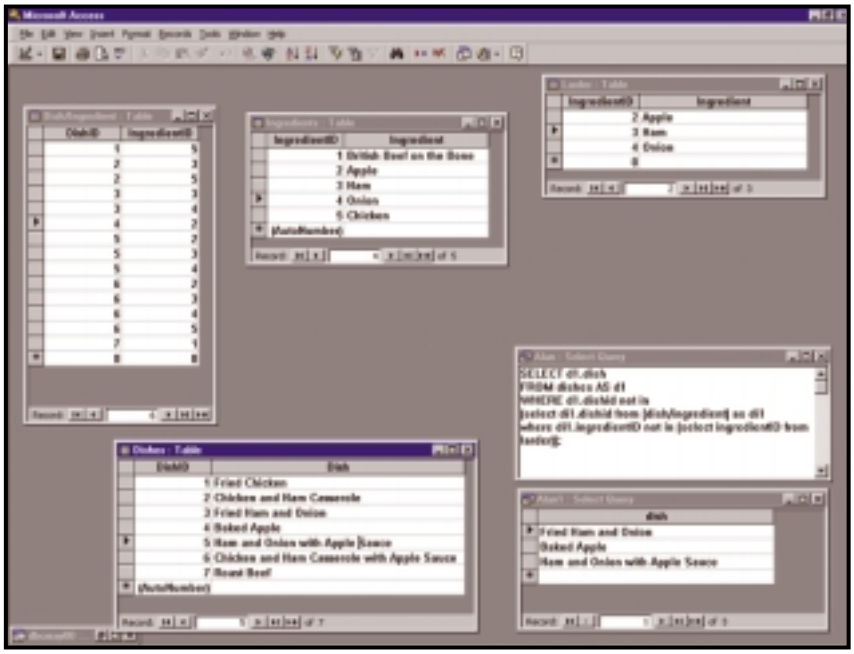
gives us the IDs of the three ingredients in the larder, which happens to be the numbers 2, 3 and 4.

So the next bit essentially reads as:

```
Select di1.dishid from ↙
[dish/ingredient] as di1
where di1.ingredientID not ↙
in (2,3,4);.
```

So it gives us IDs of the dishes that have an ingredient NOT in the larder, which, by looking at the Dishes table, can be seen to be dishes (1, 2, 6 and 7).

So the entire SQL statement now reads:



Screenshot 1: The tables used in the recipe problem/solution

```
SELECT d1.dish
FROM dishes AS d1
WHERE d1.dishid not in ↙
(1,2,6,7)
```

This returns the names of dishes (those with IDs 3, 4 and 5) which can be made.

Alan says that he doesn't mean to imply that his 'is better or that it's going to go faster or anything, it's just different'. A commendable attitude – as discussed in this column before, while speed can be vitally important, it isn't the only consideration in choosing a solution to a problem. However, it is often intriguing, so if I can find the time I'll generate a big set of data and do a bit of speed testing. You'll find the MDB file with a small data set on this month's CD-ROM as DBCMAY00. MDB for those who also want to have a play.

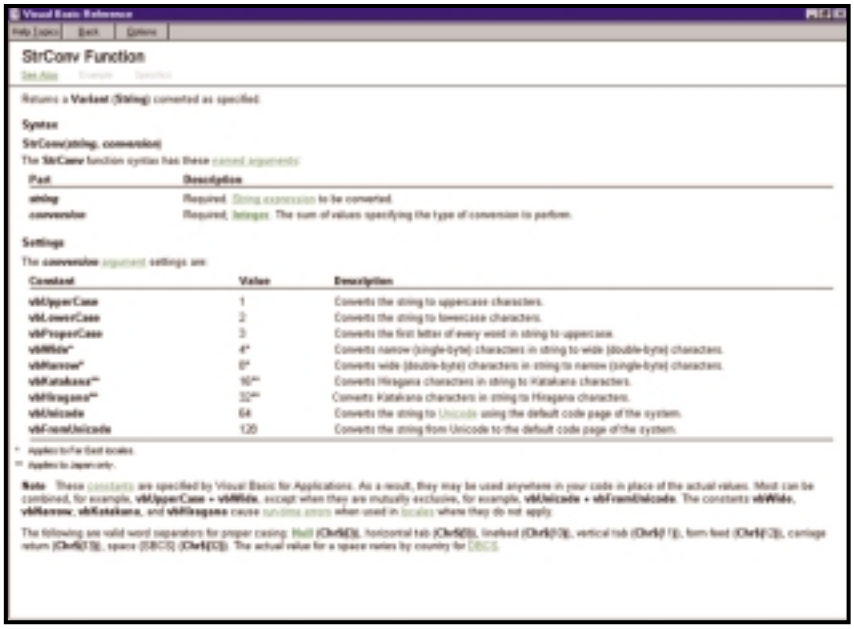
In the meantime, you might want to amuse yourself by trying to guess which will be faster; the answer may be non-intuitive. Ken Sheridan supplied the information that some tests done with DB2 in 1996 showed that relative performance of these two types of solution varies with the relative size of the tables. This tends to suggest that the choice may depend on how thick your recipe book is and how well stocked the larder. So, in order to be meaningful, any speed testing will have to be performed multiple times under different conditions. I make this point because speed is often treated in a relatively trivial way (particularly by some manufacturers!) and is typically complex.

Alan finishes his email with 'Keep up the good work, and more tricky examples please!' How do the rest of you feel? Do you want more examples of SQL and how it can be used in devious ways to entice the brain (and, as a by-product, extract data elegantly)?

If you are interested, email the address at the end of the column with a header reading 'MORE SQL'; if you find the topic boring, put 'LESS SQL'. I will count the replies and take my cue accordingly.

Just in case

Some problems can arise because data passed on to you is not in the format you require. Gary Sycamore (gary@access2data.co.uk) inherited a database from a previous employee, in which all the data has been input in UPPER



Screenshot 2: The StrConv function has a mind-boggling range of capabilities

CASE, some by the users and most from text files sent to the company from the US.

For mail merges and reports he would like to have the name, address and postcode of each entry output and printed in the proper case. He does not necessarily want to permanently change the data in the table to the proper case, although this wouldn't be a problem.

This problem sounded familiar and a little digging in the archives revealed it had been discussed back in April 1997. Then I provided an MDB file that used the StrConv function (only available in Access 95 and above) which, as the name suggests, converts strings. This function has an impressive array of switches (see screenshot 2), the one we need in this case is '3', which tells the string conversion function to perform a 'proper case' conversion. So:

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

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, etc.

For simplicity, I would update the data in the table (working on a backup copy rather than the original). This is easy to achieve using an update query: an example is included in UPCASE.MDB on the CD-ROM. Screenshot 3 shows the query and the result.

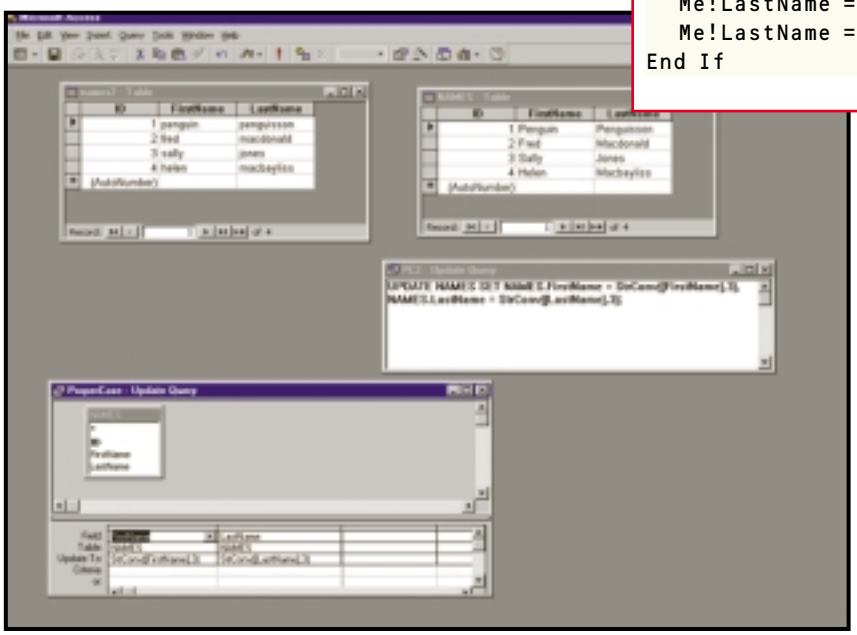
Thereafter you can rerun the query at regular intervals and/or use the sample code in the input form to capitalise the data as it is entered.

Incidentally, capitalisation is a complex issue given names such as MacDonald, etc. As you can see from the screenshot, some of the names are quite inelegantly handled. The code snippet in Figure 1 doesn't begin to be definitive, but shows how code can be used to catch the exceptions if you need to handle them.

FIG 1

Catching exceptions

```
Me!LastName = StrConv(Me!LastName, 3)
If Left(Me!LastName, 3) = "Mac" Then
   Length = Len(Me!LastName)
   Me!LastName =   Right(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!LastName
End If
```



Screenshot 3: A quick update query will fix simple case problems at a stroke

CONTACTS

Mark Whitehorn welcomes your feedback on the Databases column. Contact him via the *PCW* editorial office, or email:
database@pcw.co.uk