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!).

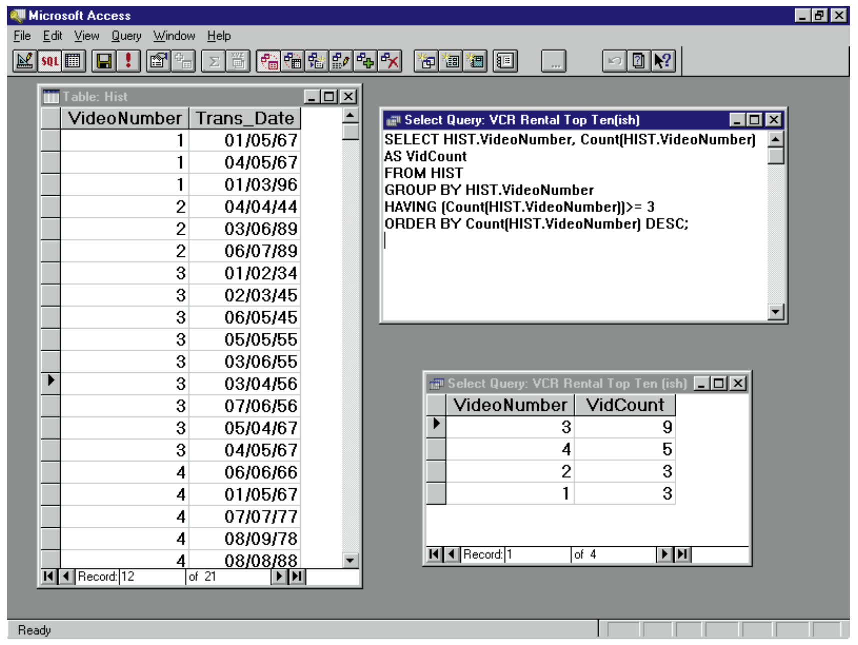
The competition went well, and was won by the Borland/Dunstan Thomas team. The tool was Delphi and the team consisted of Jason Vokes and Colin Ridgewell. Dunstan Thomas is one of Borland's Client/Server Partners based in Portsmouth which specialises in developing client/server and internet business systems.

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?


Microsoft Access
File Edit View Query Window Help
Table: Hist
Select Query: VCR Rental Top Ten(ish)
Select Query: VCR Rental Top Ten (ish)
Record: 12 of 21
Record: 1 of 4
Ready

VideoNumber	Trans_Date
1	01/05/67
1	04/05/67
1	01/03/96
2	04/04/44
2	03/06/89
2	06/07/89
3	01/02/34
3	02/03/45
3	06/05/45
3	05/05/55
3	03/06/55
3	03/04/56
3	07/06/56
3	05/04/67
3	04/05/67
4	06/06/66
4	01/05/67
4	07/07/77
4	08/09/78
4	08/08/88

```
SELECT HIST.VideoNumber, Count(HIST.VideoNumber)
AS VidCount
FROM HIST
GROUP BY HIST.VideoNumber
HAVING (Count(HIST.VideoNumber))>= 3
ORDER BY Count(HIST.VideoNumber) DESC;
```

VideoNumber	VidCount
3	9
4	5
2	3
1	3

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 ➢

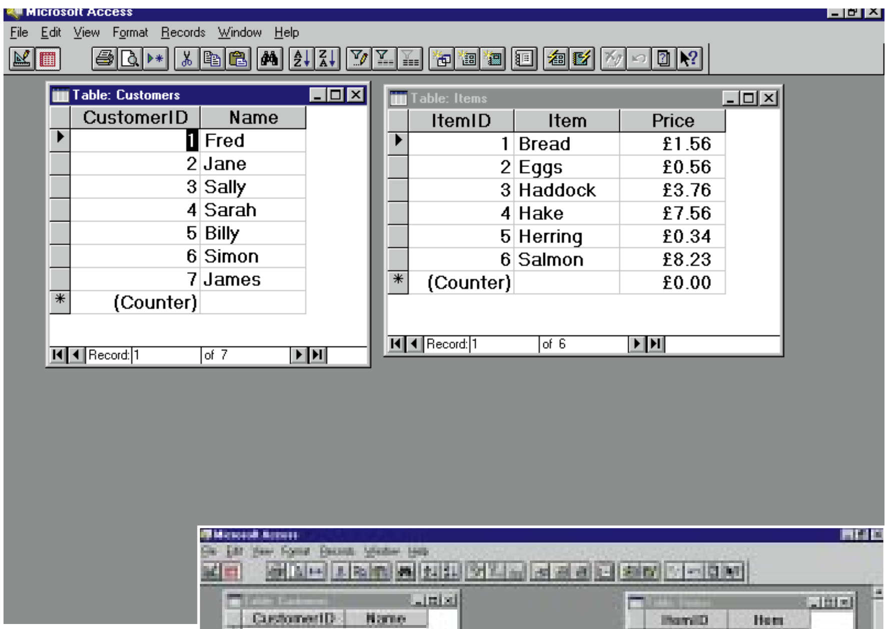


Table: Customers

CustomerID	Name
1	Fred
2	Jane
3	Sally
4	Sarah
5	Billy
6	Simon
7	James
(Counter)	

Record: 1 of 7

Table: Items

ItemID	Item	Price
1	Bread	£1.56
2	Eggs	£0.56
3	Haddock	£3.76
4	Hake	£7.56
5	Herring	£0.34
6	Salmon	£8.23
(Counter)		£0.00

Record: 1 of 6

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

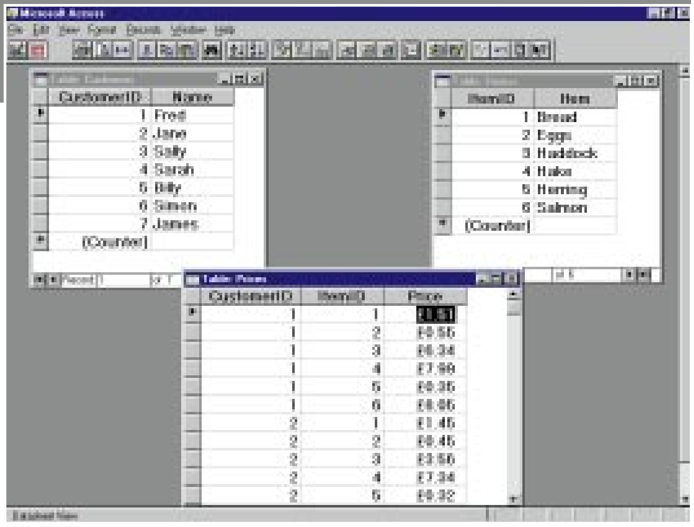


Table: Customers

CustomerID	Name
1	Fred
2	Jane
3	Sally
4	Sarah
5	Billy
6	Simon
7	James
(Counter)	

Table: Items

ItemID	Item
1	Bread
2	Eggs
3	Haddock
4	Hake
5	Herring
6	Salmon
(Counter)	

Table: Prices

CustomerID	ItemID	Price
1	1	£0.51
1	2	£0.56
1	3	£6.34
1	4	£7.99
1	5	£0.35
1	6	£8.85
2	1	£1.45
2	2	£0.45
2	3	£3.56
2	4	£7.34
2	5	£0.32

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

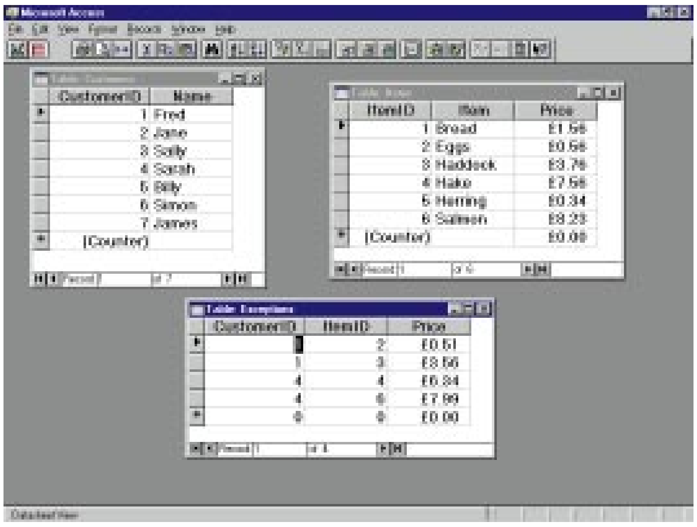


Table: Customers

CustomerID	Name
1	Fred
2	Jane
3	Sally
4	Sarah
5	Billy
6	Simon
7	James
(Counter)	

Table: Items

ItemID	Item	Price
1	Bread	£1.56
2	Eggs	£0.56
3	Haddock	£3.76
4	Hake	£7.56
5	Herring	£0.34
6	Salmon	£8.23
(Counter)		£0.00

Table: Exceptions

CustomerID	ItemID	Price
1	2	£0.51
1	3	£3.56
4	4	£6.34
4	6	£7.99
0	0	£0.00

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 ➢

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:

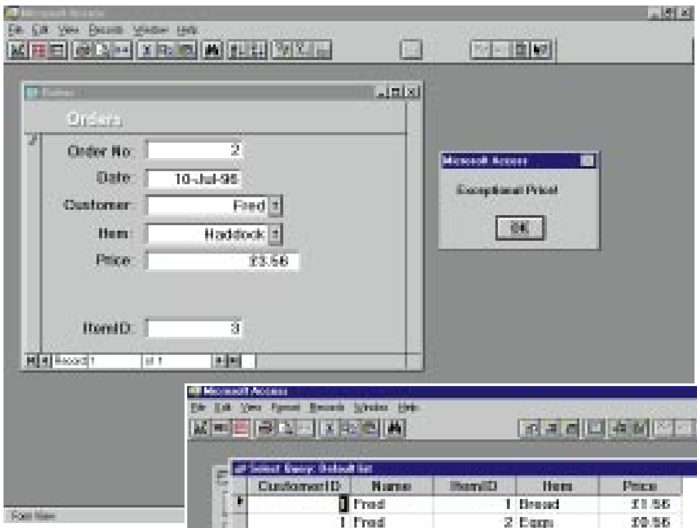
1. Open the Exceptions table.
2. Search for an entry which has this customer and this item.
3. If an entry is found, copy the price from that record into the price field on this form.
4. If not, open up the Item table, find the correct item and copy the price from there.

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?

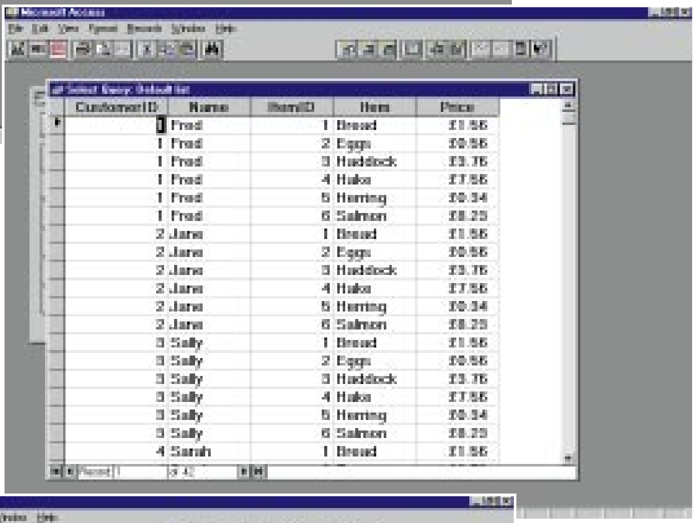

Orders

Order No:	2
Date:	10-Jul-96
Customer:	Fred
Item:	Haddock
Price:	£3.56
ItemID:	3

Exceptional Price!

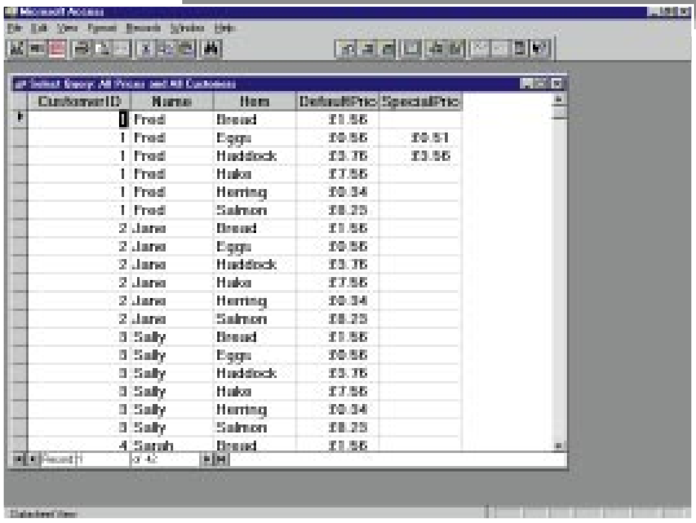
OK

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


Select Query: Default list

CustomerID	Name	ItemID	Item	Price
1	Fred	1	Bread	£1.56
1	Fred	2	Eggs	£0.56
1	Fred	3	Haddock	£3.76
1	Fred	4	Hake	£7.56
1	Fred	5	Herring	£0.34
1	Fred	6	Salmon	£8.23
2	Jane	1	Bread	£1.56
2	Jane	2	Eggs	£0.56
2	Jane	3	Haddock	£3.76
2	Jane	4	Hake	£7.56
2	Jane	5	Herring	£0.34
2	Jane	6	Salmon	£8.23
3	Sally	1	Bread	£1.56
3	Sally	2	Eggs	£0.56
3	Sally	3	Haddock	£3.76
3	Sally	4	Hake	£7.56
3	Sally	5	Herring	£0.34
3	Sally	6	Salmon	£8.23
4	Sarah	1	Bread	£1.56

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


Select Query: All Prices and All Customers

CustomerID	Name	Item	DefaultPric	SpecialPric
1	Fred	Bread	£1.56	
1	Fred	Eggs	£0.56	£0.51
1	Fred	Haddock	£3.76	£3.56
1	Fred	Hake	£7.56	
1	Fred	Herring	£0.34	
1	Fred	Salmon	£8.23	
2	Jane	Bread	£1.56	
2	Jane	Eggs	£0.56	
2	Jane	Haddock	£3.76	
2	Jane	Hake	£7.56	
2	Jane	Herring	£0.34	
2	Jane	Salmon	£8.23	
3	Sally	Bread	£1.56	
3	Sally	Eggs	£0.56	
3	Sally	Haddock	£3.76	
3	Sally	Hake	£7.56	
3	Sally	Herring	£0.34	
3	Sally	Salmon	£8.23	
4	Sarah	Bread	£1.56	

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

•PCW Contacts

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