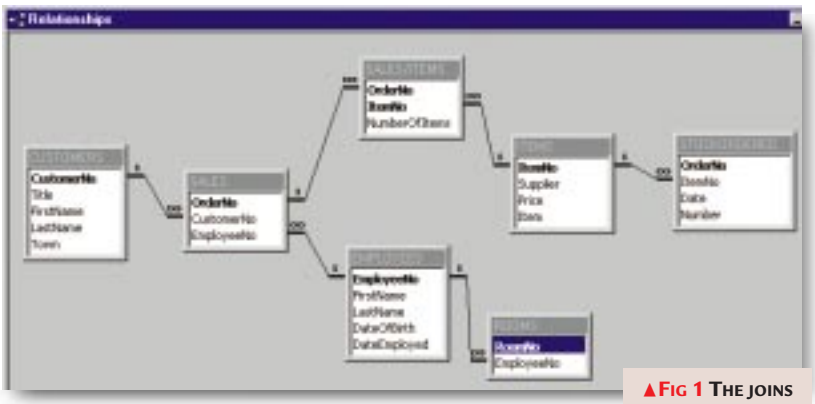
Stock around the clock

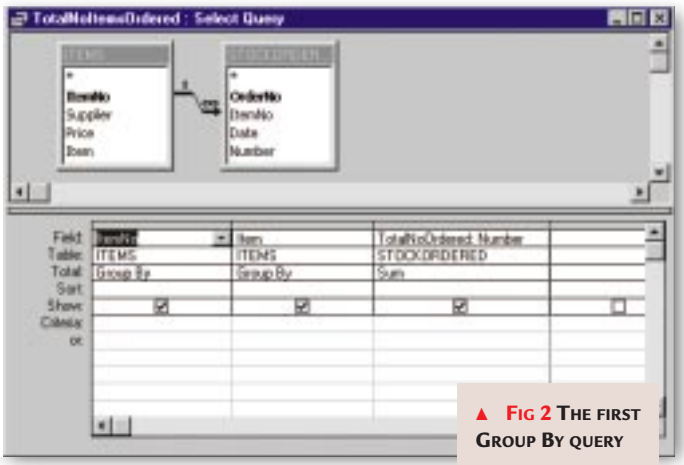
Mark Whitehorn offers a solution to the complex question of ordering systems and stock control.

An interesting question arose in an email from reader Jason Holt <Jason@creasefield.demon.co.uk> who is putting together a custom stock control system on Access 97 for his manufacturing business. He has a problem updating the 'quantity in stock' field when goods are booked in.

There are several ways of coping with stock levels, each with its own pros and cons. I don't suggest that the one I'll demonstrate is perfect for all occasions, but it does work and is a good starting point. All the following tables, queries etc., are on our *PCW* cover-mounted CD-ROM in the file DBCAPR99.MDB.



▲ FIG 1 THE JOINS BETWEEN THE TABLES



▲ FIG 2 THE FIRST GROUP BY QUERY

This database is a simple ordering system for a company that buys furniture from manufacturers and sells it on to customers. So, it has a STOCKORDERED table that holds information about its own purchases from manufacturers. It also has a pair of tables (SALES and SALES/ITEMS) which store information about its own sales of the same goods to its customers. One further table of interest in this case is the ITEMS table which simply lists the names of the items traded (Chairs, Tables and so on). Fig 1 shows the joins between these tables and also that there are further tables. These others are useful for the ordering system but do not concern us for this example.

A Group By query called TotalNoItems Ordered [Fig 2] can be used to list the total number of items sold. For non-Access users, the SQL is [Fig 3]. Another Group By query (TotalNoItemsOrdered) performs a similar operation to total up the number of items ordered [Fig 4]. Note the use of an INNER JOIN in each case to ensure that every item listed in the ITEMS table is shown in the answer table, even if the item hasn't yet been ordered or sold.

Given these two tables, it is then easy to base a third query (StockLevel) on them to produce a table of stock levels [Fig 5]. This is great and will work most of the time. But if you introduce some slightly unusual data, as I have done in the sample tables, it all goes horribly wrong.

For example, in the sample table [Fig 6] we have ordered some bookcases, but have not yet sold any. We have also sold a few chests without actually getting around

[FIG 3] Group By query SQL

```
SELECT ITEMS.ItemNo, ITEMS.Item, Sum(STOCKORDERED.Number) AS TotalNoOrdered

FROM ITEMS LEFT JOIN STOCKORDERED ON ITEMS.ItemNo = STOCKORDERED.ItemNo
GROUP BY ITEMS.ItemNo, ITEMS.Item;
```

[FIG 4] TotalNoItemsOrdered query

```
SELECT ITEMS.ItemNo, ITEMS.Item, Sum([SALES/ITEMS].NumberOfItems) AS↙
 TotalNoSold
FROM ITEMS LEFT JOIN [SALES/ITEMS] ON ITEMS.ItemNo = [SALES/ITEMS].ItemNo
GROUP BY ITEMS.ItemNo, ITEMS.Item;
```

(Key: ↙ code string continues)

[FIG 5]

StockLevel query

```
SELECT TotalNoItemsSold.ItemNo, TotalNoItemsSold.Item,↙
 TotalNoItemsOrdered.TotalNoOrdered,↙
 TotalNoItemsSold.TotalNoSold, [TotalNoOrdered]↙
-[totalNosold] AS StockLevel
FROM TotalNoItemsSold INNER JOIN TotalNoItemsOrdered ON↙
 TotalNoItemsSold.ItemNo = TotalNoItemsOrdered.ItemNo;
```

(Key: ↙ code string continues)

[FIG 6]

The answer table

ItemNo	Item	TotalNoOrdered	TotalNoSold	StockLevel
1	Desk	220	3	217
2	Lamp	50	3	47
3	Chair	0	7	-7
4	Table	2	3	-1
5	Chest		60	
6	Bookcase	500		

to ordering them. These items show up in the table but have no value in the Stock Level field, as [Fig 6] shows.

Rats! This looked simple, so why is it failing to proceed? The problem is that RDBMSes like Access are pedantically correct. When the query TotalNoOfItemsSold runs, it finds no orders pertaining to the sale of bookcases. If it returned a value of zero this would be inaccurate. The database doesn't hold the fact that zero was sold and it holds no information about how many *were* sold, so the answer that we see in the answer table is a null value.

So far so good. But the query called StockLevel subtracts this null value from a real value which is the number ordered; in this case 500 — this subtraction takes place in the bit of SQL that reads, [TotalNoOrdered]-[totalNosold]. If you subtract an unknown value from a known value, the only reasonable answer is that you don't know what the answer is. So, the StockLevel for bookcases is a null, which appears as a blank. This is not helpful and we know that since we have 500 in stock and haven't sold any, the answer should be 500.

One way is to always insert a dummy order (where the numbered ordered is zero) for every item, as I have done for Chair. This enables the calculation to proceed sensibly but this solution has problems. You will always have to remember that you have dummy orders in there, otherwise when you calculate, say, the average value per order, the answer will be wrong.

Another solution is to take the answer table [Fig 6], replace all of the nulls with zeros and then do the calculation. The next problem is that this answer table happens to be non-updatable as it provides summary values. No problem. We simply modify the query above so that it writes the answer as a 'real' table to disk. This table will then be updatable, so we can change the nulls to zeros [Fig 7]. We end up with four queries which need to be run in sequence.

- **StockLevelQuery** is a make table query that generates a table called StockLevelTable.
- **ReplaceNullsWithZero** is a query that replaces all of the null values in the TotalNoSold field in that table with zero.
- **ReplaceNullsWithZero2** does the same thing for the TotalNoOrdered field.
- **FinalStockLevel** performs the calculation of the stock level based on that table.

For non-Access users, the SQL for these queries is on the *PCW* cover CD in a text file called DBC.TXT.

FinalStockLevel : Select Query

Item	TotalNoSold	TotalNoOrdered	StockLevel
Desk	3	220	217
Lamp	3	50	47
Chair	7	0	-7
Table	3	2	-1
Chest	60	0	-60
Bookcase	0	500	500

Record: 2 of 6

◂FIG 7
THE END RESULT

QUICKIES

☛ Roger Page <roger@golant.demon.co.uk> wants to know where his Toolbox toolbar is hiding in Access 97. From the menu bar select View, Toolbars and database and it should reappear.

☛ Andrew Johnson <ajohn@hitachi-eu.com> has found that when he adds a new row containing a date field in Access 97, it generates a null value if nothing is entered in it. Andrew wonders how he can set a date field back to NULL after it has been populated with a valid date. An update query should do what he needs — see the table and query called SetDatestoNull in the database on our *PCW* CD-ROM. This update query simply looks for specific dates (in this case, anything greater than 1/1/1990) and replaces the value with a null:

```
UPDATE ANDREW SET ANDREW.[Date] = Null
WHERE (((ANDREW.Date)>#1/1/90#));
```

Running all four queries is a pain to do manually, so I have set up a button on a form called CalculateStockLevels that runs all four sequentially. It also has a useful command

```
DoCmd.SetWarnings False
```

which stops all those irritating warnings appearing telling you of the impending replacement of the data in the table. But finally, do remember that turning off warnings is dangerous.

PCW CONTACTS

Mark Whitehorn can be contacted via the PCW editorial office (address, p14) or email database@pcw.co.uk