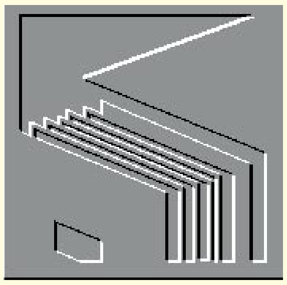


Can you see the **join**?

Mark Whitehorn has been inundated with answers to a meter reading problem.

The question I posed in the March issue has opened up a huge oyster-full of pearls. To save you having to look it up, it consisted of a table of readings from electricity meters (*Table 1).* The problem was to generate an answer table consisting of records showing each meter reading, together with the previous reading (if available) from the same meter, and to calculate information such as *units used*. The answer is shown in *Table 2*.

I concluded by stating that my colleague, Stephen, and I had "cheated" with the solution we presented in March (it works perfectly, but offends the relational model) and we both felt it was inelegant.

Last month I documented the "cheat", and now we'll look at the solution. This problem has produced the greatest number of responses I have ever received and they are still rolling in.

The responses fell into three main categories:

- Interesting, but ultimately incorrect; or correct in that they functioned but they offended the relational model in some way, just like the solution in the March issue.
- Correct; they solved the problem without offending the model.
- Pleas that the sender didn't understand the problem because he/she didn't speak SQL and wanted to know more.

In order to try and satisfy everyone (democracy at its best!) I'll publish some of the solutions, and devote part of this and future columns to a look at the basics of SQL for the benefit of those who want to know more.

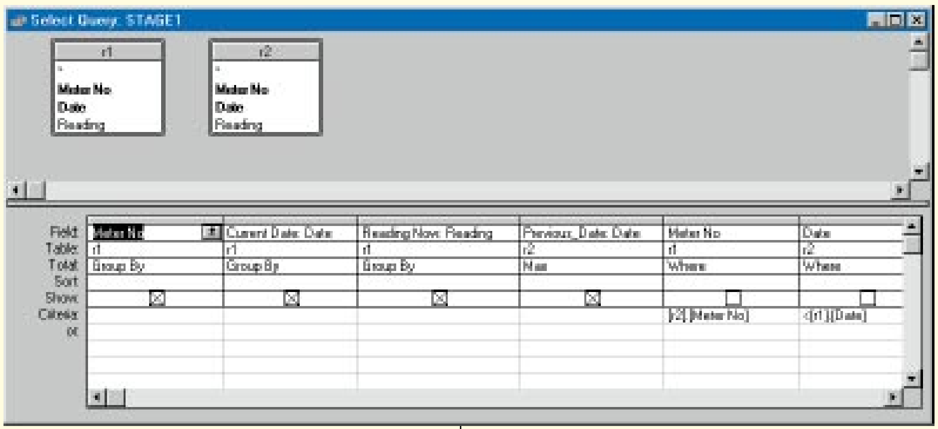
Incidentally, several people suggested that Stephen and I actually *knew* what the answer was and had simply set the question as an academic exercise! This wasn't the case. Although we eventually did come up with our own elegant solution, we hadn't

Table 1 Original readings

Meter No.	Date	Reading
1	18/05/91	20
1	11/11/91	91
1	12/04/92	175
1	21/05/92	214
1	01/07/92	230
1	21/11/92	270
1	12/12/92	290
1	01/04/93	324
2	18/05/91	619
2	17/09/91	712
2	15/03/92	814
2	21/05/92	913
2	17/09/92	1023
3	19/05/91	20612
3	11/11/91	21112
3	15/03/92	21143
3	21/05/92	21223
3	17/09/92	21456
3	21/03/93	22343

Table 2 The answer

Meter No.	Date	Current Reading	Previous Reading	Units Used
1	11/11/91	91	20	71
1	12/04/92	175	91	84
1	21/05/92	214	175	39
1	01/07/92	230	214	16
1	21/11/92	270	230	40
1	12/12/92	290	270	20
1	01/04/93	324	290	34
2	17/09/91	712	619	93
2	15/03/92	814	712	102
2	21/05/92	913	814	99
2	17/09/92	1023	913	110
3	11/11/91	21112	20612	500
3	15/03/92	21143	21112	31
3	21/05/92	21223	21143	80
3	17/09/92	21456	21223	233
3	21/03/93	22343	21456	887



Field:	Meter No	Current Date: Date	Reading Now: Reading	Previous_Date: Date	Meter No	Date
Table:	r1	r1	r1	r2	r1	r2
Total:	Group By	Group By	Group By	Max	Where	Where
Sort:						
Show:	☒	☒	☒	☒	☐	☐
Criteria:					[r2].[Meter No]	<[r1].[Date]
or:						

Fig 1 *The Access query builder for Eve Rocks' first query. Note that the self-join is essentially treating a single table as if it were present twice, and joins it to itself. Eve has used two aliases, one for each occurrence of the table. In the text I have just aliased the second table, but either approach is perfectly acceptable*

Fig 2 The select command

```
SELECT Readings.[Meter No], Readings2.date
FROM Readings, Readings AS Readings2
```

Fig 3 Using the WHERE clause

```
SELECT Readings.[Meter No], Readings2.date
FROM Readings, Readings AS Readings2
WHERE
Readings.[Meter No] = Readings2.[Meter No]
AND Readings.[Date] < Readings2.[Date]
```

solved the problem at the time of writing, for publication in the March issue.

The solutions

SQL is not as standard as many people would like, particularly with regard to the style in which statements are formatted and capitalised. In presenting these solutions I have tried, as far as possible, to leave the SQL in the exact form in which it was sent to me. However, minor modifications have occasionally been necessary in order to make the SQL work with the original table I supplied.

The basis of all the working solutions is a self-join. This is a join in which records from a table are combined with other records from the same table, when there are matching values in the joined fields. The matching values don't have to match exactly, in the sense of A = B; they can match on an expression like A > B.

The easiest way of picturing a self-join is to imagine that you can duplicate the table and then join it to itself, just as you would if they were actually different tables (*Fig 1, page 276*). In SQL, the syntax for creating a self-join is simple: you just name the table twice in the FROM clause, giving it an alias for the second version of the table (or indeed, an alias for both occurrences); thus, the select command in *Fig 2* is using Readings2 as an alias for the duplicate of the table called Readings. The join conditions can then be described using the WHERE clause as normal (*Fig 3*).

This self-join matches records where the meter numbers are the same and where the date of the second reading is earlier than that of the first.

The solution in *Fig 4* (page 278), sent in by Eve Rocks, uses exactly this type of join, although it is more detailed and uses a GROUP BY clause to group the records. From the original data, this yields the result shown in *Table 3* (page 278).

This output table is named STAGE1 in Eve's solution. It has done most of the work in that it has produced records which

Tips & Tricks: When is FoxPro like File Manager?

This bit of FoxPro code comes from Matthew Cook-McQueen: *"I wanted my FoxPro for Windows applications to operate like File Manager: you run it once and then when you double-click on the icon, the original instance is merely maximised. So I developed the code, below. It works for FPW and VFP, and maybe for the Mac if that supports DDE (but not for the DOS version):*

```
*----- (This goes at the start of the main program in your app)
=ddesetoption("SAFETY",.F.)
ch = ddeinitiate("MyApp","SYSTEM")
IF CH != -1
     =DDETERMINATE(CH)
     QUIT
ENDIF
DO DDE_SETUP

PROCEDURE DDE_SETUP
     =DDESETSERVICE("MyApp","DEFINE")
     =DDESETTOPIC("MyApp","SYSTEM","DETECTED")
     RETURN

*----- (This bit can go in your procedure file if you have one)
FUNCTION DETECTED
PARAMETERS A,B,C,D,E,F
    ZOOM WINDOW SCREEN MAX
    RETURN .T.
```

"The theory is that your application is set up as a DDE server. When you run your application, the first thing it does is to see if it can connect to this server. If it can, the function DETECTED is run and the original application is maximised. Control is then returned to the second instance of the application, which quits itself. If the application cannot connect to the server, it 'knows' that it is not already running, so it sets up the server."

Fig 4 Eve Rocks' solution

```
STAGE1:
SELECT DISTINCTROW r1.[Meter No], r1.date AS current_date, r1.reading AS reading_now, Max(r2.date) AS previous_date
   FROM readings AS r1, readings AS r2
WHERE r1.[Meter No] = r2.[Meter No] AND r2.date<r1.date
GROUP BY r1.[Meter No], r1.date, r1.reading;
```

Table 3 Data resulting from Eve Rocks' solution

Meter No.	Current Date	Reading Now	Previous Date
1	11/11/91	91	18/05/91
1	12/04/92	175	11/11/91
1	21/05/92	214	12/04/92
1	01/07/92	230	21/05/92
1	21/11/92	270	01/07/92
1	12/12/92	290	21/11/92
1	01/04/93	324	12/12/92
2	17/09/91	712	18/05/91
2	15/03/92	814	17/09/91
2	21/05/92	913	15/03/92
2	17/09/92	1023	21/05/92
3	11/11/91	21112	19/05/91
3	15/03/92	21143	11/11/91
3	21/05/92	21223	15/03/92
3	17/09/92	21456	21/05/92
3	21/03/93	22343	17/09/92

contain information from two records in the original table. Not only that, it has managed to find the correct records to join together. Note that there are 16 records in this table and 19 in the original; this is because three records (the first one for each meter) do not have a preceding reading.

You may be wondering why Eve hasn't simply pulled the rest of the data we need (such as the Previous reading) into the table at the same time. The answer is that using a GROUP BY clause restricts the fields that you can actually display in the answer table.

However, the bulk of the work is done and the SQL statement (*Fig 5*) joins this table back to the original one (Readings) and pulls in the missing information that we want.

This yields the correct answer table (apart from minor differences, such as the field names) as shown at the start of the column in *Table 2*.

Many people sent in solutions like this which used two SQL statements: some were similar to Eve's; others used different means to achieve the same ends. Nothing in the relational model excludes the use of multiple SQL statements; indeed, using two consecutive statements like this makes the solution easier to understand.

Fig 5 SQL statement

```
STAGE2:
SELECT DISTINCTROW readings.meter, stage1.current_date,
readings.date AS previous_date, stage1.reading_now,
readings.reading AS old_reading,
stage1.reading_now - readings.reading AS units_used
FROM readings INNER JOIN stage1
ON readings.meter = stage1.meter
AND readings.date = stage1.previous_date;
```

Fig 6 Peter Davidson's solution

```
SELECT DISTINCTROW Readings.[Meter No], Readings.Date,
Readings.Reading AS [Current Reading],
Readings_1.Reading AS [Previous Reading],
[Readings].[Reading]-[Readings_1].[Reading] AS [Units Used]
FROM Readings, Readings AS Readings_1
WHERE ((Readings.[Meter No]=[Readings_1].[Meter No])
AND (Readings.Reading>[Readings_1].[Reading])
AND (((SELECT count(*) from Readings AS i
WHERE i.[Meter No] = Readings_1.[Meter No]
AND i.[Date] > Readings_1.[Date])-(SELECT count(*) from Readings AS j
WHERE j.[Meter No] = Readings.[Meter No]
AND  j.[Date] > Readings.[Date]))=1))
ORDER BY Readings.[Meter No], Readings.Date;
```

Single file

However, complete solutions can be generated as a single SQL statement, as in this one from Peter Davidson (*Fig 6*).

As he says: *"The use of a self-join in this way is very common. It is perhaps surprising how often data is related to itself in one way or another. The result of a self-join is the Cartesian product of the table with itself and, as with any type of join, it is necessary to restrict the rows only to those that make sense. In my statement, that is achieved in the first part of the WHERE clause which projects only those rows where both meter numbers are the same.*

"The second part of the WHERE clause selects only those rows that have a current reading date which is greater than the previous reading date. This is not entirely necessary but it has a use, especially for tables with lots of rows. The advantage is that it cuts down the number of rows presented to the final, processor-intensive part of the WHERE clause."

From the Oracle

Peter wasn't the only one to send in a complete solution as one SQL statement, and not all worked in the same way. *Fig 7* is in Oracle (ANSII) SQL, which isn't too surprising because it comes from Tony Willis-Culpitt, principal consultant with the Oracle Corporation. Tony provided a detailed description of the working of the statement, which included the following information:

*"The function around the date just turns it into a more readable format. The column alias' (in double quotes) just give nice headings to the SQL*Plus columns.*

"In order to add the previous readings to each of the records returned, construct a second set on the same table where, for each of the meter numbers, the read_date is the next below the one in the current record. Unfortunately, this has to be done as a correlated sub-query but careful indexing should mean that it is still fairly efficient.

Fig 7 Tony Willis- Culpitt's statement

```
select   r1.meter_no "Meter No",
     to_char(r1.read_date,'dd-Mon-yyyy') "Reading Date",
     r1.reading "Reading",
     r2.reading "Last Reading",
     r1.reading - r2.reading "Units Used",
     to_char(r2.read_date,'dd-Mon-yyyy') "Last Read"
   from          meter_readings r1,
     meter_readings r2
   where         r1.meter_no = r2.meter_no
    and r2.read_date = ( select max(r4.read_date)
                         from      meter_readings r4
                         where     r4.meter_no = r1.meter_no
                         and       r4.read_date < r1.read_date)
  UNION
  select         r5.meter_no,
      min(to_char(r5.read_date,'dd-Mon-yyyy')),
      min(r5.reading),
      0,
      0,
      null
    from         meter_readings r5
    group by     r5.meter_no
    order by     1, 2 ;
```

"The change to the order by clause is because the column names cannot be used explicitly in a union query. 1 and 2 are the relative positions of the sort columns in the select list. The 'min' function on the 'r5.reading' column does not actually do anything positive; it is just there to satisfy the rule that any returned column not in the 'group by' clause must have a group operator applied to it."

And thanks to...

It seems grossly unfair to single out only the few who have been mentioned so far. The respondents below each deserve a mention; either for correct answers, or for just plain interesting ones. My solution and the best of the rest are in the magazine directory on this month's free, *PCW* CD-ROM.

Malcolm Bacchus	Alasdair Macdonald
Paul Bloomfield	Alan Mackechnie
Jose Femenias	Charles Mawdsley
Adrian Fowle	John Meads
David Gould	Brian Riley
Dave Johnson	Geoffrey Snook
Andrew Kaye	

The speed angle

You may remember that this question first appeared, in the March issue, with reference to speed. The problem *can* be solved with sequential programming in Access Basic, but Stephen and I were keen to find an SQL answer for reasons of speed as well as elegance.

A code solution that we produced for testing purposes is horribly slow, taking several seconds to produce an answer for a single meter. The SQL answers are essentially instantaneous, producing answers in well under a second. So, that answer is clear — use SQL whenever possible.

The quest continues

I am now interested to know whether there is a difference between the single and multiple SQL solutions. I know that the answer is going to depend on the optimiser which Access uses, and that any differences I detect may well reflect differences other than the number of SQL statements. But I'm still going to construct a mega-table of meter readings and do some more speed tests.

Finally, in case this sounds like a diatribe against using an RDBMS's internal programming language (in this case, Access Basic) I am quite happy to use the language to generate the large block of data I need for testing. SQL is not a full programming language; it is a subset of one. It is excellent (and very fast) for querying databases, but useless for other things. Once again, it's a case of horses for courses. ▣

PCW Contacts

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