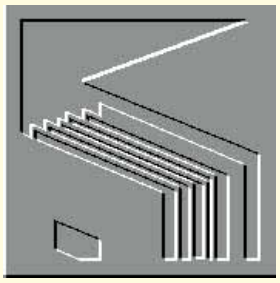


meter matching machines

Mark Whitehorn examines a problem thrown up by a meter reading database, and sings the praises of indexing — this month looking at how it can speed up data retrieval.

Last month I started to write about ways of speeding up your database. This month I will digress slightly to cover a problem which has arisen since then. A colleague of mine, Stephen Elwell-Sutton, was building a database in Access to store meter readings from various electricity meters. These can be stored in a table like that in *Fig 1*:

Fig 1 Meter readings

Meter No	Date	Reading
1	18/5/91	20
1	11/11/91	91
1	12/4/92	175
1	21/5/92	214
1	1/7/92	230
1	21/11/92	270
1	12/12/92	290
1	1/4/93	324
2	18/5/91	619
2	17/9/91	712
2	15/3/92	814
2	21/5/92	913
2	17/9/92	1023
3	19/5/91	20612
3	11/11/91	21112
3	15/3/92	21143
3	21/5/92	21223
3	17/9/92	21456
3	21/3/93	22343

using [Meter No] and [Date] as the primary key. The people using the database wanted, quite reasonably, to see the data in the sort of format shown in *Fig 2*, from which other information, such as the days between readings, and the usage per day, can be calculated.

Each record in this table is made up of data taken from two records in the original table, and the relationship between the two records concerned in each case is as follows. Consider the last record in the original table to be the "current" record. In order to generate a record in the second table we scan backwards, looking for the first record which has the same meter number and a date which is lower in value that the current record.

Fig 2 Another format

Meter No	Date	Current Reading	Previous Reading	Units used	Date of Previous Reading
1	11/11/91	91	20	71	18/5/91
1	12/4/92	175	91	84	11/11/91
1	21/5/92	214	175	39	12/4/92
1	1/7/92	230	214	16	21/5/92
1	21/11/92	270	230	40	1/7/92
1	12/12/92	290	270	20	21/11/92
1	1/4/93	324	290	34	12/12/92
2	17/9/91	712	619	93	18/5/91
2	15/3/92	814	712	102	17/9/91
2	21/5/92	913	814	99	15/3/92
2	17/9/92	1023	913	110	21/5/92
3	11/11/91	21112	20612	500	19/5/91
3	15/3/92	21143	21112	31	11/11/91
3	21/5/92	21223	21143	80	15/3/92
3	17/9/92	21456	21223	233	21/5/92
3	21/3/93	22343	21456	887	17/9/92

Given that the data is sorted as shown in the first table, this will be the record above the current one. This matching process fails if we consider the record for meter three taken on the 11/11/91, since the record above it is for meter two.

So, on the face of it, the algorithm for solving the problem was trivial:

- Start at the bottom of the table.
- Repeat
- For each record look at the one above. If it contains an identical meter number, use the two records to generate a record in the answer table.
- Move up one record, until at the top of the table.

In fact, you can work from the top to the bottom in a similar manner if you prefer. Fine. Except that the solution is inelegant, if not to say offensive.

Why? Well, one of the major principles of the relational model is that the position of a record in a table is of no significance whatsoever. We are supposed to be able (and indeed willing) to locate the value in a field solely by reference to its table name, field name and primary key value. Just because a value in a field happens to be in "the record above the current one" doesn't mean we automatically know it's the one to use. SQL, for example, doesn't allow you to reference the records above or below the current one; indeed SQL has no concept of the "current" record. Instead it performs operations on sets of records rather than on individual ones.

Typically, the programming language supplied with most RDBMSs will allow sequential record processing, and either Stephen or I could have solved the

Fig 3 Solving the problem with SQL

1.

```
DELETE *
FROM Readings2
```

2.

```
INSERT INTO READINGS2 ( [Meter No], [Date], Reading )
SELECT [Meter No], [Date], [Reading]
FROM READINGS
ORDER BY READINGS.[Meter No], READINGS.[Date]
```

3.

```
SELECT DISTINCTROW [Meter No], Date, Reading, [Count]+1 AS [Incremented
Count]
FROM READINGS2
```

4.

```
SELECT DISTINCTROW READINGS2.[Meter No], READINGS2.Date,
READINGS2.Reading AS [Current Reading], [3 Renumber records in a
dynaset].Reading AS [Previous Reading], [Current Reading]-[Previous
Reading] AS [Units used], [3 Renumber records in a dynaset].Date AS [Date
of Previous Reading], [READINGS2].[Date]-[3 Renumber records in a
dynaset].[Date] AS [Days since last reading], [Units Used]/[Days since
last reading] AS [Daily Usage]
FROM [3 Renumber records in a dynaset] INNER JOIN READINGS2 ON ([3
Renumber records in a dynaset].[Incremented Count] = READINGS2.Count) AND
([3 Renumber records in a dynaset].[Meter No] = READINGS2.[Meter No])
```

problem with Access Basic, but instead we looked for an entirely SQL-based solution. And in case you think this problem has wandered into the realms of academic theory, remember that set operations have the potential to be *much* faster than sequential processing, so we were looking not only for a more elegant solution, but a faster one.

The solution we came up with is as follows:

The data was stored in a table called READINGS. We created a table with an identical structure to READINGS, except that it had an additional field called [Count] which was of type Counter. This table was called READINGS2.

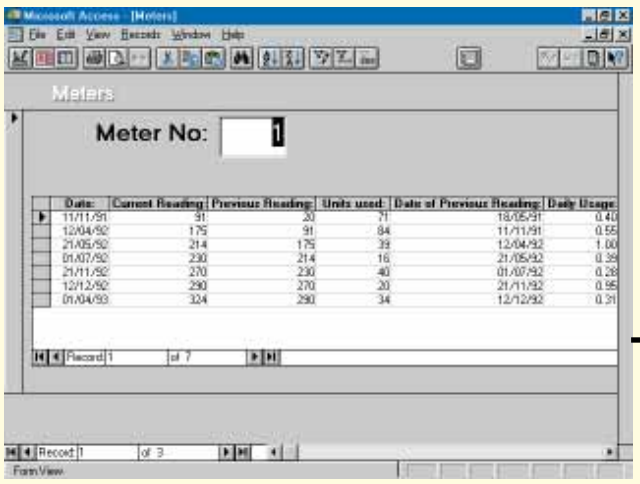
(As an aside, it is a characteristic of Access that if you add data to a table which has a counter field, Access will automatically increment the counter field, sequentially numbering the records. If the table has been used before, the numbers won't start at 1, but they will still be sequential.)

Then we wrote a series of SQL statements which did the following:

1. Cleaned out any existing data from READINGS2 .
2. Copied the data from READINGS to READINGS2, keeping the order identical to that dictated by the primary key.
3. Created a Dynaset in which the values in the counter field were incremented by one.
4. Joined READINGS and the Dynaset by Meter No and the Counter to produce an answer table which then contained the desired results.

The important parts, and hopefully the rationale, should become clear if you study the sample data.

The SQL was as shown in Fig 3.



A master/sub form which presents the "meter" data in a more acceptable manner

The table names are overlong, I agree, but this means they make more sense in the data which is on the cover disk as METER2.MDB.

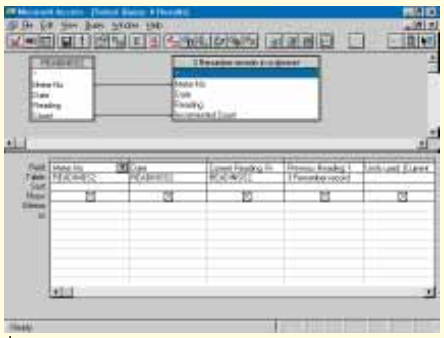
So, we completed the task we set ourselves, and solved the problem with SQL. Hopefully this is faster than sequential processing would be. Nevertheless we are unhappy with the result, because we cheated. Somewhere in there is a nasty, cheap little trick which offends the relational database model.

Three questions arise.

1. Can you find the cheat?
2. Can you do better?
3. Are we missing something vital?

It seems to us that this problem is representative of a broader class of problem: "What is the best way of handling records in a relational database that have to reference each other?"

Clearly there are other solutions, such as storing both the current and previous reading in each record, but they all seem to conflict with the relational model as well. Surely someone, somewhere has worked out a solution to this in the past? Anyone out there know what we are missing?



The GUI version of the final SQL statement

PCW Contacts

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

Tips & Tricks

From Shane Devenshire:
Selecting an entire form or report without using the menus in Access

When you are in the Form or Report design mode of Access, you often want to select the entire form or report. To do this you can click the white box in the top left corner of the form or report. Alternatively, if you can see any space to the right hand edge of the form or report all you need do is click it. Whatever you click to the right of — the Detail section, Page Header — they all work. If you can't see that area you can click in the area below the Report Footer; this will also select the entire form.

In Access II you can also click or double click anywhere on the ruler that has no objects directly below. Double clicking on the horizontal ruler will bring up the form's Property dialogue box.

From Damian Luby

I am looking after a database which belongs to the estate I live in. I would like to have the primary index (and sorting) put the address in proper sequence.

I have the address in the following format:

1 Glenbourne Road
2 Glenbourne Road
11 Glenbourne Road
12 Glenbourne Road

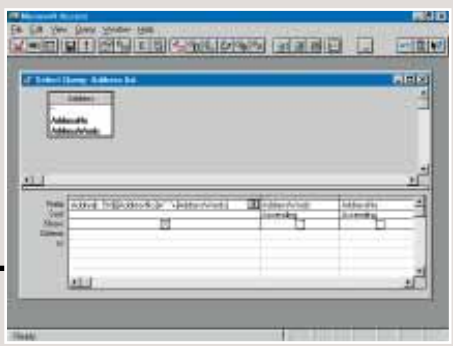
After sorting, the no 11 house is put before the no 2, naturally enough. Can you think of an easy work-around for this problem? I am using Microsoft Access 2.0.

Try splitting the address into two fields:

AddressWords
AddressNo

and declare the field called AddressWords to be of data-type Text and AddressNo to be of type Number.

Access allows you to combine the two fields to form a single primary key, so that you can make AddressWords AND AddressNo the joint Primary key. The fields will then sort properly.



The query which lists addresses in a tidy, sensible fashion

To which Damian replied:

Thanks, but how can I combine the house no. & address field to be sorted when using queries in Access 2.0, so the addresses come out in order?

Build a new query and in the first column write:

```
Address: Str$([AddressNo])+" "+[AddressWords]
```

The function Str$ turns the numeric value in AddressNo into text and the rest of the expression simply adds that text value to a space character and then to the text in AddressWords. This causes values like:

```
7 Acacia Gdns.
```

to appear in the first column of the answer table. However, sorting on this column will be unsatisfactory because Access will treat the numeric values as text, which means we are back to the original problem. Clearly what we want is the addresses sorted first by street name, then by number, as in the addresses table. In order to achieve this, add the two fields AddressWords and AddressNo to the answer table (in that order) and set each in turn to be sorted in ascending order.

Addresses

1 Acacia Gdns.
7 Acacia Gdns.
12 Acacia Gdns.
13 Acacia Gdns.
17 Acacia Gdns.
24 Acacia Gdns.
241 Acacia Gdns.
2 Belmont Road
6 Belmont Road
7 Belmont Road
12 Belmont Road
56 Belmont Road
214 Belmont Road

Since AddressWords is the first sorted field in the answer table, reading from left to right, the data will be sorted first by the values in that field. The data in the AddressNo field will be used to sort data with identical values in the AddressWords field. However, since AddressNo contains numeric values, the sort will proceed correctly.

Finally you can render AddressWords and AddressNo invisible in the answer table by deselecting the "Show" option. Even deselected in this way, they will still be used to sort the data. For non-Access users, the SQL for this query is:

```
SELECT DISTINCTROW Str$([AddressNo])+"
  "+[AddressWords] AS Address
  FROM Address
ORDER BY AddressWords, AddressNo;
```