One is not enough

A reader worries that as the work gets too big for his company's current database system to handle, which way now? Mark Whitehorn is on hand to dispense this, and other, advice.

I would like a discussion of the relative merits of a client/server database and a networked database application in which there is a server. When working on database applications, I am always considering small standalone systems for one PC. Most of our work so far has been in-house and thus manageable. As time goes by and we are involved in increasingly big projects, I am beginning to worry that the one-PC approach will fall on its face.

I believe I have two options:

1). Put my data on a big server running something like NT and run multiple copies of my Paradox application all pointing at the tables on the server. Then let Paradox handle the problems. I realise I'd have to be careful about record locking and the like.

2). Up-size my data to a client /server application using something like Interbase running on a server (UNIX?) and do it all through SQL, although my knowledge here is very hazy."

Alasdair Macdonald

I received this email and it seems a broad enough question, to warrant some expansion. After all, it is one of the biggest decisions that you are likely to make, and is an area where mistakes are both common and expensive. There are essentially four database models you can employ:

1). Everything on a standalone PC.

2). PC front-end — data on file server.

3). Client server using a database server as the back end.

4). Mainframe

The fourth seems inappropriate for discussion in a PC magazine, so we'll ignore it. What I can do here is to outline the strengths and weaknesses of each of the other three, together with approximate performance and size estimates.

This is a complex area. The decision to choose one of these alternatives will be based upon the interaction of many factors, including response time required, number of users, file size, file number, data size, available resources (including hardware, software and money) and type of data access required (read only, read write).

These factors interact in complex ways. Suppose that your system definition makes multi-user access to the data essential: you can instantly rule out a standalone PC. If the number of users is guaranteed to be small (say, three), then on a given hardware platform you could allow those users to access a greater volume of data than if there were 50 of them. If the number of users did then increase, the system might still work, but the response time would drop.

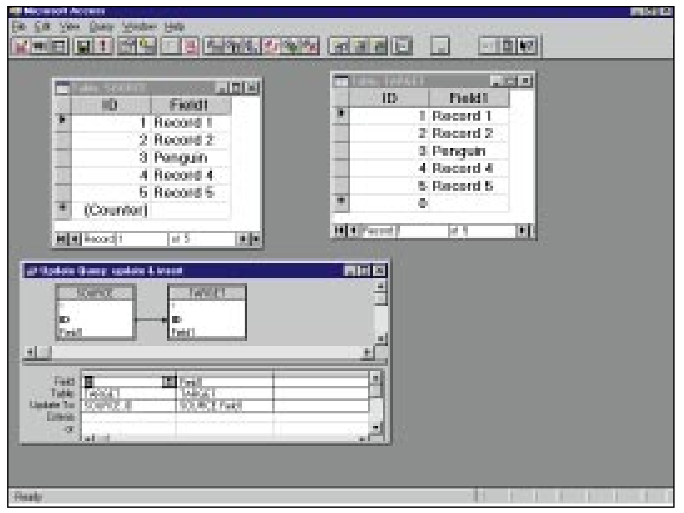


Table: SOURCE

ID	Field1
1	Record 1
2	Record 2
3	Penguin
4	Record 4
5	Record 5
(Counter)	

Table: TARGET

ID	Field1
1	Record 1
2	Record 2
3	Penguin
4	Record 4
5	Record 5
0	

Field:	ID	Field1
Table:	TARGET	TARGET
Update To:	SOURCE.ID	SOURCE.Field1
Criteria:		
or:		

(Fig 1, top right) The update query described by Paul [see page 276]. The two tables are the same

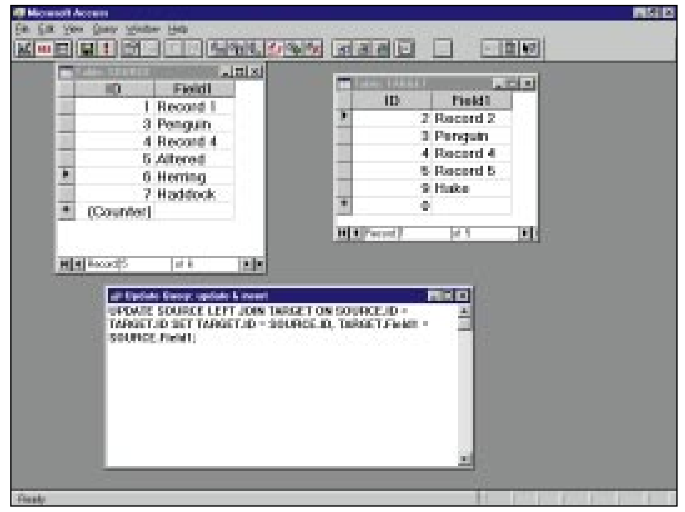


Table: SOURCE

ID	Field1
1	Record 1
3	Penguin
4	Record 4
5	Altered
6	Herring
7	Haddock
(Counter)	

Table: TARGET

ID	Field1
2	Record 2
3	Penguin
4	Record 4
5	Record 5
9	Hake
0	

```
UPDATE SOURCE LEFT JOIN TARGET ON SOURCE.ID =
TARGET.ID SET TARGET.ID = SOURCE.ID, TARGET.Field1 =
SOURCE.Field1;
```

(Fig 2, right) Here I have made changes to both of the tables, but have yet to run the query. (I have also altered the view of the query to show it in SQL)

To make matters worse, the interactions between these factors are often non-linear: for example, doubling the number of users on a given system might have little impact on response time. Doubling it again might bring the same system to its knees.

It's easier not to give any actual figures, but this is likely to leave you gnashing your teeth and wondering "What exactly counts as 'lots'? Three? Twenty five? Five hundred?" On the other hand, if I do quote hard figures, like saying that you shouldn't consider using a standalone PC for more than 1Gb of data, there'll be someone out there happily using a 200MHz P6 with 1024Mb RAM to access 1.5 Gb.

I will quote figures because it seems far more useful to do so, but please just regard them as general figures from which to start discussions. Please don't take them as gospel, and please don't build your entire database strategy around them alone.

Work alone on a standalone

The simplest database model is to install everything on a standalone PC. You use an RDBMS like Access, Paradox, dBase and FoxPro to manipulate the data.

Only one person can use it at a time, and I wouldn't use this sort of system for more than about 1Gb. Factors which affect this figure are the hardware (more memory equals larger data files) and the manner in which the data is used. For example, if it's rarely updated, then it can be heavily indexed and queries should run against it reasonably rapidly. If the data is constantly updated, the indices will slow down the updating, and yet removing them will slow down the querying! In a nutshell, if the data is rarely updated, heavily indexed, and you have very impressive hardware, you can go above this limit. With a 286 with 640Kb, don't even think about it.

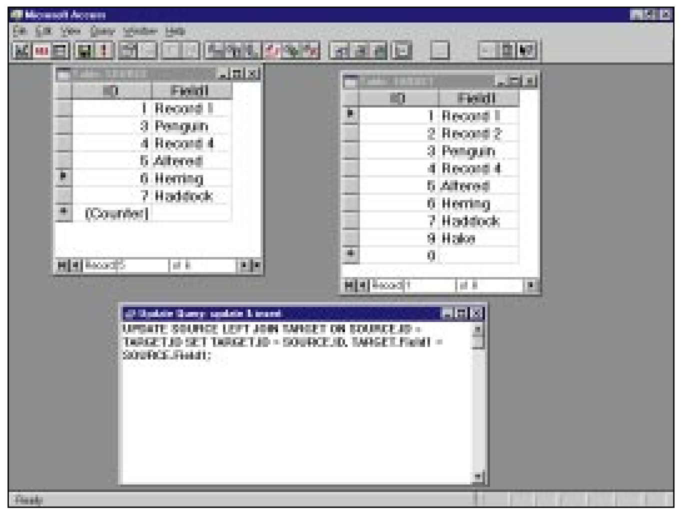


Table: SOURCE

ID	Field1
1	Record 1
3	Penguin
4	Record 4
5	Altered
6	Herring
7	Haddock
(Counter)	

Table: TARGET

ID	Field1
1	Record 1
2	Record 2
3	Penguin
4	Record 4
5	Altered
6	Herring
7	Haddock
9	Hake
0	

```
UPDATE SOURCE LEFT JOIN TARGET ON SOURCE.ID =
TARGET.ID SET TARGET.ID = SOURCE.ID, TARGET.Field1 =
SOURCE.Field1;
```

(Fig 3) The state of the tables after the query has been run

The major advantage of this sort of system is that it is cheap, and easy to manage. A database can be thought of as four different parts:

1). User interface section.

2). Data processing engine.

3). Conflict resolution section (to deal with conflicts introduced by multiple users accessing the data at the same time).

4). The data itself.

In a standalone PC-based database there is only one user, so the conflict resolution section isn't required and the other three appear as a single, seamless entity to the user. Simple. In fact, it is *so* simple, why would you ever want to go to anything more complex?

One of the major reasons for moving to a more complex database model is that this one cannot handle multiple users. For one thing, there is only one keyboard, so we can expect fist-fights if we try for multi-user. Also, this model doesn't allow for conflicts between the requirements of different users to be resolved. If you need more than one user to access the data at the same time, it is time to split up the components described above and partition them between different machines. This leads us to the second database model, which we'll cover in the next issue.

Target, aim, fire!

"Suppose we have one table, called Source, containing new records, and another table called Target. We wish to insert into Target the records in Source that are not already in Target, and update those records which are already in both tables to

p276 ➢

Database Systems
by Paul Beynon-Davies

This book looked promising. Many of the subjects covered are of interest, and the style, while a little formal for my tastes, is perfectly respectable. However, reading it in more depth reveals a series of unnerving flaws. For a start, the book is heavily cross-referenced. I like cross-references, but I do like the references to point to the correct place. Far too many here do not. Exactly half of the cross-references in chapter six, for example, are incorrect. With a failure rate like that, they are too frustrating to use. And it is not only the cross-references that are flawed.

The same chapter covers SQL and the author demonstrates retrieval, ordering and grouping for which he uses a base table, eight SQL samples and eight answer tables. Amazingly, three out of eight answer tables are incorrect, a state of affairs likely to induce severe confusion in a novice reader.

This book is aimed at students, but cannot be recommended to them or anyone else, which is sad because in many ways it's a fine book. It is simply crying out for a careful revision. Hopefully the next edition will be improved, but steer clear of this one.

MacMillan Press ISBN 0-333-63667-8. £19.99

the values of the records as in Source.

Instinctively, programmers will achieve this by two queries:

1). Update

2). Insert

In Access it can be achieved with a single Update query with a LEFT JOIN.

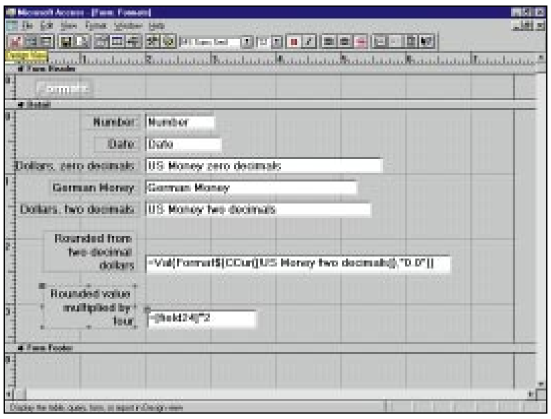
```
UPDATE SOURCE
LEFT JOIN TARGET ON
SOURCE.ID = TARGET.ID
SET TARGET.ID = SOURCE.ID,
TARGET.Field1 = SOURCE.Field1;
```

Note that both tables have fields called ID and Field1, and both are of the same data dimensions in each table. Remember that if ID is a counter field in Source, it must be a long integer in Target. This works, since Access matches those records in Source which are not in Target with a Null Record in Target which can then be updated. From an SQL point of view, this technique may not work in other DBMS but is jolly useful in Access 2.0 and Access 7.0.

I enclose a sample Access 2.0 database [on the cover disk as pdbdemo.mdb] which demonstrates this. To see it in action, play around with appending/changing records in Source and then run the query to see the effects in Target. The uses of this technique are numerous, and variations on the query to supply selection criteria make it powerful."

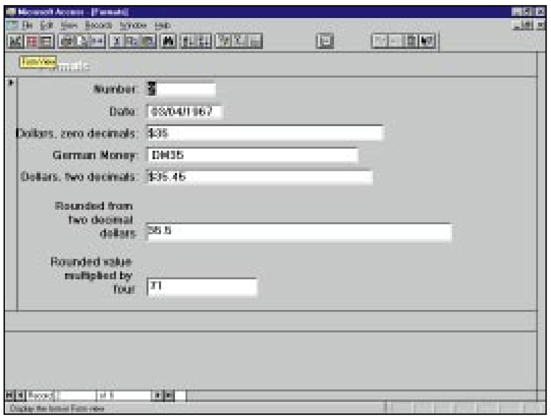
Peter Blackburn

It is worth noting that this query will not delete records from Target which have been deleted from Source. This is not a criticism. If it did delete those records, it would effectively be replacing Target with a copy of Source. It is simply a characteristic of this type of query. I can't help feeling that this might help in the solution of last month's problem concerning Mark Squire's problem about Customers and Items.



Number:	Number
Date:	Date
Dollars, zero decimals:	US Money zero decimals
German Money:	German Money
Dollars, two decimals:	US Money two decimals
Rounded from two decimal dollars	=Val(Format$(CCur([US Money two decimals]),"0.0"))
Rounded value multiplied by four	=[field24]*2

(Fig 4, right top) **Here I have used some of the formats suggested by Paul, and his rounding mechanism**



Number:	8
Date:	03/04/1967
Dollars, zero decimals:	$35
German Money:	DM35
Dollars, two decimals:	$35.46
Rounded from two decimal dollars	35.5
Rounded value multiplied by four	71

(Fig 5, right bottom) **And here is how the data appears. Notice that the value shown by the rounding mechanism is still a number; it can be mathematically manipulated as shown in the lowest "text" box. This form is also in the sample database on the cover disk**

Currency codes: help wanted

"Your July column covered the formatting options for dates in Access.

It is often overlooked that the Format property in Table, Query, Form and Report design is not restricted to just those formats on the list. The variety of codes available is the same as those used in Excel's Format, Cells, Number dialogue box.

Thus, a code of dd/mm/yyyy will show dates with century, $#,##0 will show amounts with dollar signs, "DM"#,##0 will show amounts in deutschmarks. This latter is especially useful since Access picks up the default currency format from Control Panel. It can then be overridden to show different currencies on a single Form, etc.

The ability to override the currency format has an additional benefit. Since currency fields are held to a fixed four decimal places internally, they are likely to be more precise than single or double numbers which can give puzzling rounding errors. As you know, Access has no exact equivalent to the =Round function in Excel. Currency fields used for other than currency can be formatted as #,##0, or #,##0.00 if two decimal places are required.

Incidentally, to get round the lack of an =Round equivalent, I use the Format$ function. This converts numbers to text, but rounds properly as we know it. With a representative sample of nearly 3,000 records, the following nesting of functions gave correct rounding when calculating VAT:

```
RoundNo = Val(Format$(CCur
(Number), "0.00"))
```

where RoundNo is the result and Number is the number or calculation to be rounded. Do you know of a better way of doing this?"

Paul le Gassick

The answer is that I don't. Anyone else? (See Figs 1-5.)

PCW Contacts

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