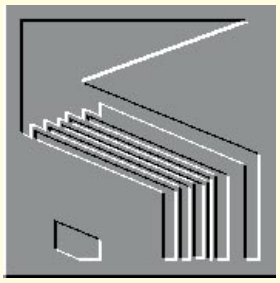


# **Speed** is of the essence

**If database users need indices to keep speed up, Mark Whitehorn asks why 70 percent of databases don't maintain an index? Plus Query optimisation, and using the Count function**

### WordArt

Microsoft WordArt is a nice little add-in that lets you embed nifty font attributes into your programs. If you own any Microsoft products, you probably have this program on your computer.

To insert a WordArt object:

**1**. Choose Edit, Insert Object in Access (or click the Object Frame Tool on the Toolbox toolbar).

**2**. Depending on your version, choose Create New and select Microsoft WordArt2 (or if you only have 1.0, choose that, and click OK).

**3**. Type your text. Note the toolbar and menu is Word Art's.

**4**. Click the first dropdown list, "Plain Text", and choose the flow you would like your text to have. (You probably can see the effect of this in the background as you work.)

**5**. Choose Format, Stretch To Frame (or use the 7th tool from the right) so that the text will fit in its frame if you resize it back in your program.

**6**. Choose Format, Shadow and pick a shadow style and colour if you want one (or click the second button from the right).

**7**. Choose Format, Border or click the first button on the right to choose a border colour and thickness for the text.

**8**. To rotate or stretch the text, choose Format, Rotation And Effect or click the fourth button on the right. With this option you can increase or decrease the amount of stretch, as well as rotate your text.

**9**. If you want to colour your text or add a pattern to it, choose Format, Shading or click the third button from the right.

**10**. To return to your program, simply click out of WordArt.

**11**. To edit the WordArt object, just double click it and you will be back in Word Art.

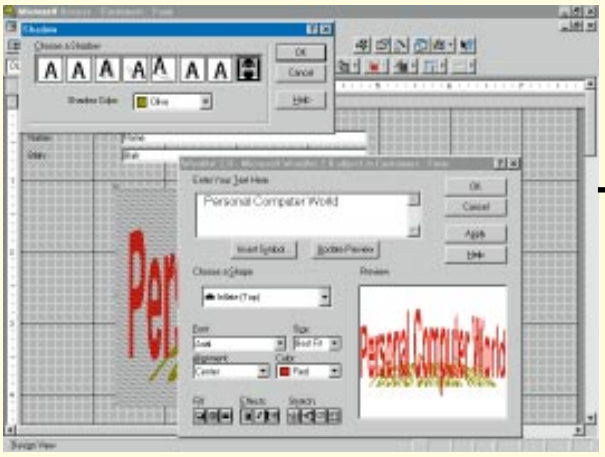
### Paradox pause

The fastest way to get out of an operation in Paradox is to press Ctrl+Break (Ctrl+Pause). Fast it may be, but it's also very dangerous. In most cases in Paradox, pressing Ctrl+Break will end a task and return you to the main menu, but nothing you were doing will be saved. Here are some of the places you can use this command: to exit the edit mode without saving changes; to exit the sort screen without executing a sort; to exit the Modify Restructure screen without saving changes; and to exit the Create mode without saving your work.

Ctrl+Break is particularly dangerous in the Report Writer. When you are in the Output to screen mode, the result of pressing Ctrl+Break depends upon where you are. If you are not at the end of the report, it brings you to the end of the current page; the second time you press it, you go to the end of the report; the third time, you return to the report writer. If you mistakenly think you're not at the end of the report and press Ctrl+Break twice, you will exit the output screen and then the report writer without saving any changes. Take heed.

### The need for speed

Microsoft recently looked at a whole collection of databases "from the wild" — databases that real users were running in a cross-section of real companies, not just from the



**Left** *Using WordArt to create a text object for use in Access*

**Below** *The WordArt object embedded in the form. Restrained and dignified, I hope you'll agree*

Fortune 500. One surprising fact that emerged was that about 70 percent of those databases had no indexing whatsoever (presumably apart from the Primary keys). Since indices are a vital component in keeping a database running rapidly, this may well account for users' frequent complaints about the tardiness of their databases.

Access 95 now boasts the ability to create indices for you automatically (as do some other RDBMSs), and this is to be applauded. However, it is still worth knowing why an index should make such a difference to data retrieval speed. Back in the December issue I promised to cover the general topic of speed, so here we go.

Speed is one of a range of factors (including usability, data integrity and scalability) which are all-important when choosing an RDBMS. Speed isn't the be-all-and-end-all, but it *is* pretty important. Speed is influenced by a very large number of factors which include (not in order of priority):

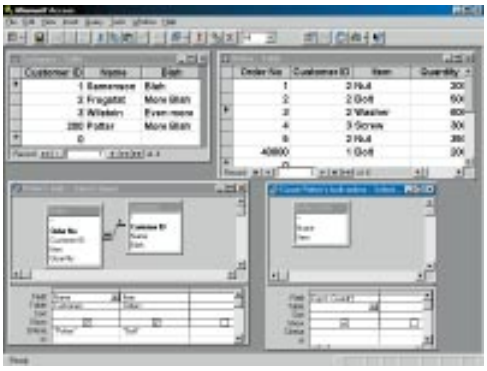
- hardware (speed of processor, amount of memory, etc.);
- disposition of data across the hardware;
- query optimisation;
- size of tables;
- number of tables;
- disposition of data within tables; and
- indexing

I don't intend to digress into a hardware discussion here, but the most common advice I hear is: "Buy more memory". The disposition of data across hardware platforms is all bound up with the machines themselves.

### Query optimisation

Query optimisation is more germane, since it is under software control. It is important in consideration of speed, because not all requests for the same information are identical. It is possible to express several requests for exactly the same data in a variety of different ways. Despite the fact that the

## Count me in



*Using nested queries to ensure that Access can count right down to zero. The two tables are shown at the top of the screen. The query on the left counts the number of times Potter has bought bolts (zero). The query on the right counts the number of records produced by the first query (zero as well). The result of this second query will display properly on a form*

Well, I asked for tips and tricks, and here is one in the form of a question from Stuart Elliot, which he rapidly followed with the solution.

"I am trying to use the Count() function to return the number of records that match the selection Criteria in a selection Query. The function normally returns the correct number, when at least one record matches the Query's selection Criteria. However, when no records match this Criteria, the function returns an #Error code. The Count function is being used in a "text box" on a sub-report, based on the aforementioned Query. Is there any way I can get the Count function to return a zero (i.e. "0") when no records match the Query's selection Criteria?" Before I could answer this, the following arrived:

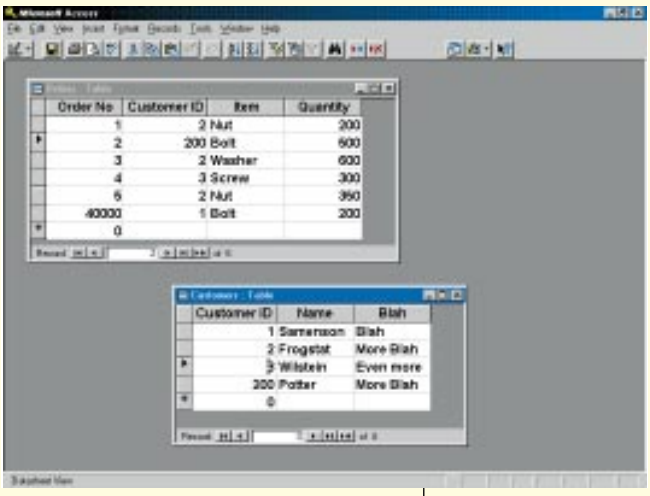
"I now have an admission to make. I found the answer to my problem, and in the MS Access User Guide at that [he says, embarrassed. No shame in that: how many of us actually read all the manuals? — MW] The information is on page 130 of the User Guide for Access v1.1.

The answer is to create a second query that contains the first query. Here are the steps:

1. Create a new selection query.
2. "Add" the first query to the second (instead of a table).
3. In the second query's first "Field" cell, enter "Count(*)" without the quotes.
4. Select "Save As" for the second query and give it a meaningful name.

How it works: The first query selects a record set according to its selection criteria. The second query generates a total of the number of records returned/selected by the first query. The * in the Count function means count ALL records in the record set (returned by the first query, in this instance), even those with Null fields."

I love problems that solve themselves.




*The two tables used in the discussion on optimisation*

**Fig 1 Nuts and bolts**

| Customer ID | Name | Blah |
|---|---|---|
| 1 | Samenson | Blah |
| 2 | Frogstat | More Blah |
| 3 | Wilstein | Even More |
| 200 | Potter | More Blah |

and their orders which number 40,000:

| Order No | Customer ID | Item | Quantity |
|---|---|---|---|
| 1 | 2 | Nut | 200 |
| 2 | 200 | Bolt | 500 |
| 3 | 2 | Washer | 600 |
| 4 | 3 | Screw | 300 |
| 5 | 2 | Nut | 350 |
| 40000 | 1 | Bolt | 200 |

answer table will be the same in all cases, the queries can take wildly different times to run.

If this sounds impossible, consider the example in *Fig 1* (page 314). Suppose you have a table of 200 customers (you'll have to imagine the missing 196 and 39994 records respectively).

Assume that we only sell four items — nuts, bolts, washers and screws — and that they all sell equally well.

Suppose you were performing the search manually and wanted to find all of the orders which have been placed by Potter for Bolts. There are several strategies you could adopt:

You might look through the 40,000 records in the ORDER table and extract every record which shows a sale of bolts — approximately 40,000 / 4 = 10,000 records. Then you could search through the CUSTOMER table, and find out that Potter has [Customer ID] 7. Finally, you could extract from your table of 10,000 records all those which have [Customer ID] = 7; perhaps 500 records.

But, of course, you wouldn't really do this. You would search through the CUSTOMER table, and find out that Potter has [Customer ID] 7, and then you would look through the 40,000 records in the ORDER table and extract every record which had [Item] = "Bolt" and [Customer ID] = 7.

Why would you do it that way? Because the first strategy searches through a total of 50,200 records, the second only 40,200; so the second should be (simplistically) about 20 percent faster.

This is a simple example of optimisation, and most RDBMSs would handle it without any trouble. Many queries of multi-table databases are much more complex than this, in particular those which involve queries which are nested to several levels. For example: "Of the employees who have dealt with client X within the last six months and sold her item Y, how many have sold product Z in the last three months to clients in countries with a GNP greater than that of country A, and who have a bad credit rating with more than two of our credit sources?"

An optimiser should be able to arrange for this request to be processed efficiently. However, the optimiser has to do more than simply decide the order in which the tables are processed. For example, in a multi-user system, an optimiser might receive several queries at about the same time, which require a specific table to be queried in the same way. Clearly it is more efficient to simply query this table once and "share" the resulting answer between the incoming queries.

Query optimisers are complex entities, and you are unlikely to want to write your own, so you might wonder why I have included them in a list of factors which you can look at to improve performance. The crucial point is that not all optimisers are equally efficient; so the choice of optimiser can have a significant effect on speed.

## What it is to be normal

The size and number of tables and the way the data is spread among them, can clearly have a marked effect upon performance. However, the interaction of these effects can be complex.

Consider a badly designed database where little or no attempt is made to normalise the data. It contains a small number of tables, each stuffed full of duplicated data; result — a glue-like response. Now consider a properly normalised database. Lots of small, neat tables, no duplicated data; result — a glue-like response.

What? They can't both be bad, can they?. Let's consider why each of these (admittedly extreme) examples might be slow.

In the first, non-normalised example, the tables are "artificially" large because of all the repeated data, so the RDBMS has to look at huge numbers of records in order to find the ones you seek.

In the second example, the tables don't suffer from duplicated data, but there are so many tables that the RDBMS spends an inordinate amount of time opening and closing them, checking the data dictionary for join information, maintaining referential and data integrity, that it has little time left for the users and their queries.

This is not to say normalisation is a waste of time. In general it's wonderful, and should certainly be regarded as the default choice. Like all tools though, it doesn't have to be used blindly.

## Two's company

In a database which is used by several users, some entering data and some querying the database, those who are querying it are generally managers, trying to see the historical trends within the data. The problem is that the "management" queries are large and complex, and interact badly with the inputting work. Essentially there are two databases needed here — one which accepts data input, and the other which is a data pool or "data warehouse". Once we make that momentous decision, we should also be able to see that the need for normalisation differs between the two.

More on this, and indexing, next issue.

**PCW Contacts**

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