Making the distinction

Mark Whitehorn on the peculiarities of DISTINCTROW in Access. Plus, if you're developing a large application, see how code and components can be versatile and non-specific.

I've written this before, but it bears repeating. This column has evolved to a state where it is almost exclusively about Access. When I write about general issues (normalisation, SQL, and so on) I try to keep it as generic as possible but the fact remains that, essentially, all the questions I receive (far more than appear on the page) refer explicitly to Access.

There are probably two reasons. Either everyone who reads this column is using Access, or people who are using another RDBMS read the column and think "Huh. That's all about Access. No point in writing in about anything else."

I'm happy to accept any suggestions for the future direction of topics covered and platforms used. So if you want changes please let me know and minority interests can be represented.

Distinctly exact (or exactly DISTINCT)

In my series about SQL (*Hands On Workshop, PCW Oct'96-Jan'97*) I said that "the generic DISTINCT becomes DISTINCTROW in Microsoft's Access". By that I meant that if you build a query in Access using the GUI, then by default it will generate an SQL statement which starts:

```
SELECT DISTINCTROW….
```

In many cases this will return the same record set as an SQL statement which starts:

```
SELECT….
```

or even

```
SELECT DISTINCT….
```

The differences between SELECT and SELECT DISTINCT are explained in my workshop series on SQL, and since that was to do with generic SQL it didn't seem to be the time or place to go into the peculiarities of DISTINCTROW. But this seems like an excellent place, so we will do so! In order to demonstrate this we need a couple of tables:

CLIENTS

ClientID	Name	Location
1	Fred	Wellington
2	Sally	Wimpy
3	Jean	Halifax
4	Fred	Lancaster

ORDERS

OrderNo	ClientID
1	1
2	4
3	4
4	4
5	3
6	3
7	3
8	1
9	1

Note that we have two clients called Fred and that Sally has yet to place an order with our company.

Both

```
SELECT Name
FROM CLIENTS;
```

and

```
SELECT DISTINCTROW Name
FROM CLIENTS;
```

return four records:

Name
Fred
Sally
Jean
Fred

whereas

```
SELECT DISTINCT Name
FROM CLIENTS;
```

returns

Name
Fred
Jean
Sally

DISTINCT (as explained in my SQL Workshop) forces SQL to remove the duplicates in the answer table.

In this case (and in all cases where a single table is queried) there is no difference between SELECT and SELECT DISTINCTROW. However, if we bring two tables into the query, the two forms can be distinguished.

```
SELECT Name
FROM CLIENTS INNER JOIN ORDERS
ON CLIENTS.ClientID =
ORDERS.ClientID;
```

returns:

Name
Fred
Fred
Fred
Fred
Jean
Jean
Jean
Fred
Fred

which is one record for each order in the ORDERS table because the join between the tables has essentially generated nine records, of which we have asked to see only the name of the customer. By contrast,

```
SELECT DISTINCT Name
FROM CLIENTS INNER JOIN ORDERS
ON CLIENTS.ClientID =
ORDERS.ClientID;
```

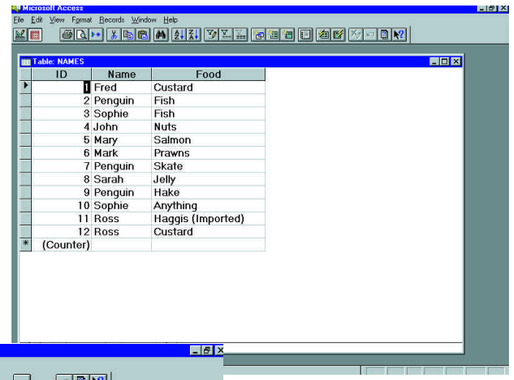
returns only two records:

p276 ➢

Name
Fred
Jean

The SELECT statement on its own returns nine records, then the DISTINCT part removes the duplicates.

The "problem" with this answer table is that it implies that there are only two people placing orders, whereas we know it is three because there are two people called Fred. If we expand the SQL statement to include the location field,



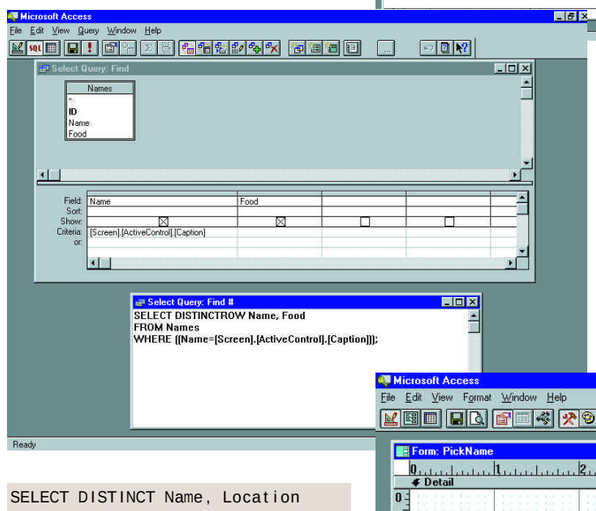


Fig 1 (above) The names used for the button-caption example

Fig 2 (left) The query called Find, as both GUI and SQL

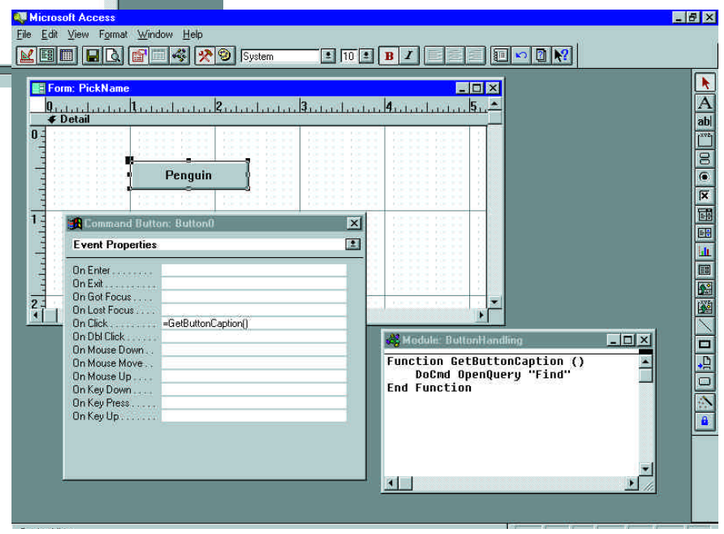


Fig 3 The form called PickName and the associated function which calls the query called Find

```
SELECT DISTINCT Name, Location
FROM CLIENTS INNER JOIN ORDERS ON
CLIENTS.ClientID = ORDERS.ClientID;
```

then we get:

Name	Location
Fred	Lancaster
Fred	Wellington
Jean	Halifax

DISTINCT is very literal; it returns unique records in the answer table, whether or not they come from unique records in the original table.

I used the term "problem" above, but of course this is only a problem if you don't know what DISTINCT is supposed to do. In fact, DISTINCT is doing exactly what it was designed to do.

My guess is that Microsoft felt that naïve users of a database might not appreciate this level of subtlety. If they saw a single name in an answer table, they would expect that it represented a single person. So the default in Access was set to DISTINCTROW which in this case, as you will by now have guessed, produces a separate record in the answer table for each unique customer who has placed an order.

Thus:

```
SELECT DISTINCTROW Name
FROM CLIENTS INNER JOIN ORDERS
ON CLIENTS.ClientID =
ORDERS.ClientID;
```

returns

Name
Fred
Jean
Fred

By the way, these tables and queries are all on our cover-mounted disc as DISTINCT.MDB.

■ I'll leave you with this brainteaser: *If you include the primary key value from CUSTOMERS in all of these queries, does the difference between SELECT DISTINCT and SELECT DISTINCTROW disappear?* As they say in all the best text books, explain your answer!

Hundreds and thousands

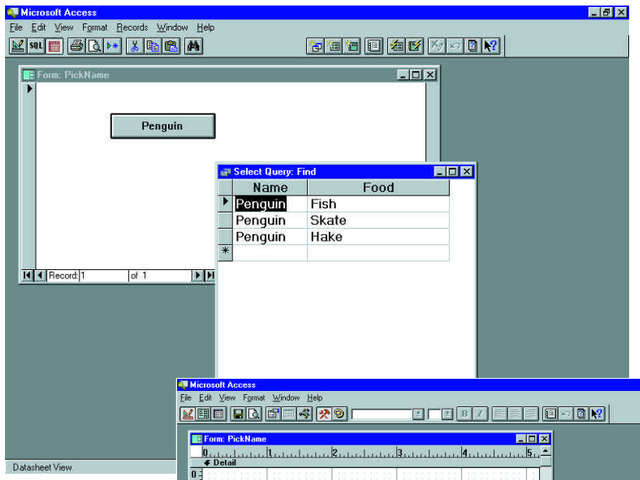
Reader, Glenn Rowe, recently contacted me with an interesting database problem: "*I'm developing an application which stores data about many countries. The data is stored in a table, one field of which stores the country code (GB for Great Britain… and so on). The interface will contain many buttons; one for each country. Each button will have the country code as it's caption and, when pressed, will return the records for that country. (These are, of course, arranged hierarchically on many forms, not all on one!)*

The problem, as far as I can see, is that I'm going to need hundreds/thousands of queries and as many snippets of code. Building it will be a nightmare, as will debugging and maintenance. Surely this process can be centralised in some way?"

The question is a good general one about the way in which code and components can be versatile and non-specific. The obvious solution, at first, is to try to pass the button's caption to the query as a parameter.

As far as I know (and I stand to be corrected) this cannot be done. However, we can achieve exactly the same result by building a query which snatches the caption from the button which has just been pressed. In other words, you can't attach a piece of code to the OnClick event property of a button which says "Run a query and pass to it the caption of this button". Instead, you get the button to run a query and get the query to locate the caption of the button which has just been pressed.

In order to do this, we can make use of an object called "Screen" and one of its properties, "ActiveControl". These are defined in the manual thus: the Screen object refers to a particular Microsoft Access form, report, or control; the ActiveControl Property is used to refer to the control that has the focus. Or, to put that another way, Screen.ActiveControl always points to the control which has the focus.



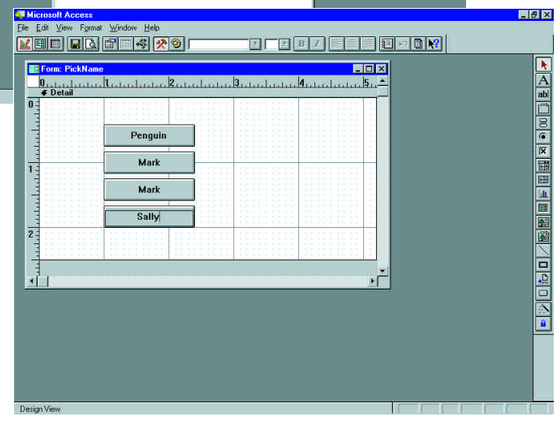


Fig 4 The answer table which appears when the button labelled Penguin is pressed

Fig 5 Cloning the button

Now consider a button on a form; if you press it, that button, by definition, has the focus. If the button runs a query which has Screen.ActiveControl.Caption as its criterion, then Property will return the caption of the button.

The beauty of this scheme is that when you design the query, you don't have to know which button is going to be pressed. You don't even need to know which form the button will be on. As long as the button calls up the query when it is pressed, the query will seek out the button, read its caption, and use that caption as a criterion.

To demonstrate this, I have used people rather than countries. This choice simply reflects the fact that I don't know enough country codes to fill even a sample table.

The table shown in the screenshot Fig 1 contains names of some individuals and the food they like to eat. The query called Find (Fig 2) has, in the criteria line:

```
[Screen].[ActiveControl].[Caption]
```

The form called PickName (Fig 3) has a single button with the caption "Penguin". The OnClick property of this button is set to

```
=GetButtonCaption ()
```

which is the name of a function. The

p278 ➢

Round and round we go

In the November issue, I published the following correspondence from Paul le Glassick: "*…Incidentally, to get around 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."

Paul wanted to know if there was a better way. Simon Hawkins suggested:

```
RoundNo = int (( Number * 100) +1) /100
```

"*Whether this is a faster method than using the Format function, described in the article, I am not sure. Also, the above may need altering to deal with negative amounts. Hope this is of some use.*"

This has the great advantage of elegance. However, when I tested it in the form

```
Function Simon (Number) As Integer
   Simon = Int((Number * 100) + 1) / 100
End Function
```

it returned 1 from 1.49 (which seems right), but 2 from 1.4999 (which seems wrong), and then 2 from 2.49999 (which seems right), and even 2 from 2.499999999 (which is still right, but conflicts oddly with the result from 1.4999!). In other words, it is slightly inconsistent. Or maybe it's my 486 processor. In the form:

```
Function Simon2 (Number) As Double
    Simon2 = Int((Number * 100) + 1) / 100
End Function
```

it returns 1.51 when given 1.5 (which seems wrong).

This reply came from James Talbut: "*I don't like converting things to and from strings. There is an operator in VB that appears to round correctly, and that is the '\' operator (integer division). Making use of this in a function is simple:*

```
Function Round(dNumber As Double, iNumDigits As Integer) As Double
Dim dFactor As Double
  dFactor = 10 ^ iNumDigits
  Round = ((dNumber * dFactor) \ 1) / dFactor
End Function
```

"This function is simple, quick, and produces the same results as the version using Format$ that you published. Interestingly it gives a different result to that of the ROUND function in Excel. For some reason Round(2.15, 1) does not give 2.2 (as it would in Excel), it gives 2.1 as does the formula you published."

All of these suggestions work, up to a point, but none are perfect (see the form called "Rounding" in Fig 6). So the plot thickens. Anyone got any further thoughts?

By the way, just as we were going to press, Simon came back with: "*OK. Classic case of using my memory instead of looking the code up (and testing it!). The function should read*

```
TxtOutput =
Int((TxtInput *
100) + .5) / 100
```

"This will round to two decimal places. Sorry about the earlier confusion."

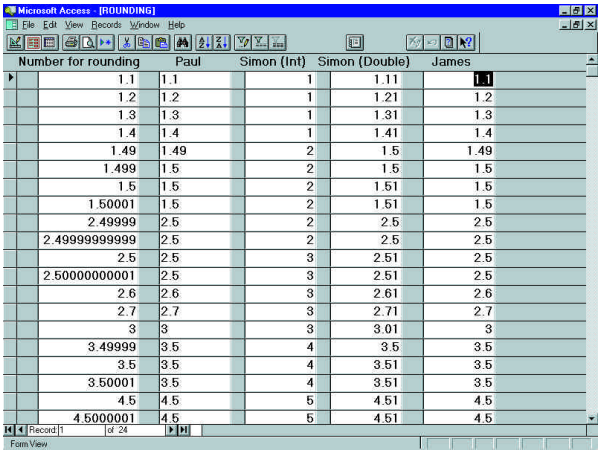


Fig 6 The rounding functions in operation. Note the purely fortuitous grouping of biblical names!

function, also shown in Fig 3, is composed of a single line which simply calls the query.

So, the steps are simple: when you press the button, the function is called; the function runs the query; the query looks for the active control (which is the button), captures its caption, and uses that caption as a criterion. The result is the answer table shown in Fig 4.

To keep it short and sweet, there is no error trapping in the demonstration code. It is also rather crude. For example, unless you close the query, pressing a different button won't re-query the table. However, all of this can be cured by expanding the code in the function from its current, rather minimalistic, one line.

Having answered that question, I realised that, essentially, the solution relied on the fact that the function could be called as soon as the button was pressed. This meant that the solution couldn't be applied to a situation where several selections needed to be made. I know there are manifold ways of handling multiple selections and that combo boxes are often the best solution, but I was intrigued to see if an economical solution (in terms of code and number of queries) could be found while retaining the button-caption-snatching idea.

The form MultipleSelections shows this in operation. A little of the elegant simplicity of the earlier solution is lost but much can be retained. Each button runs a function which places the button's caption in the text box above it. This text box is explicitly named in the function, so each button can be cloned vertically but each separate column of buttons is calling a separate function. Thus, if we made ten selections, we would need ten functions. However, by using Screen.ActiveForm (a close relative of Screen.ActiveControl) in the functions, a degree of flexibility has been retained so that forms which are cloned from this one should also work, as long as the same names are used for the text boxes.

The "Go" button fires a query called (rather imaginatively, I feel) "Find2". The query looks at the text boxes in the form, takes the values from there and runs the query. By using Screen.ActiveForm this query should be versatile enough to work with different forms.

PCW Contacts

Mark Whitehorn welcomes readers' correspondence and ideas for the Databases column at database@pcw.vnu.co.uk

p000 ➢