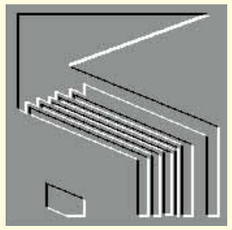


Onto a **winner**

Mark Whitehorn **considers information storage, data dictionaries and catalogues as part of his ongoing examination of Codd's rules, and the National Lottery provides an interesting problem.**

Codding about

Continuing with the list of suggested rules for modern, PC-based RDBMSs, we have:
8. An RDBMS must maintain a data dictionary for each database which stores information about the joins between the tables, referential integrity, etc. Access to the tables which circumvents this data dictionary should be forbidden.
A "data dictionary", also known as a "catalogue", is a centralised store of information about the database. It contains information about the tables: their number, names, the fields they contain, data types, primary keys, indices and so on. However, it should also contain information about the joins which have been established between those tables: foreign keys, referential integrity, cascade update, cascade delete and so on.

Incidentally, this usage of the term "catalogue" should not be confused with its use in dBase, despite the fact that both usages are misspent in the same way. In dBase, a catalogue is simply a container for all components of the database: data files, queries, forms, programs and such like. There is no storage of information about the joins between the tables.

One of the major functions of a true data dictionary is to enforce the constraints placed upon the database by the designer, such as referential integrity and cascade delete. In the early days of the PC, none of the "relational" DBMSs offered a true data dictionary, but this wasn't a major concern for two reasons. Firstly, the early PCs were very slow and incapable of manipulating large, complex, multi-table sets of data. Instead they tended to be used for fairly simple, single-table work (address lists, for example) so the deficiencies in the DBMS didn't show up as much as they might otherwise have done. Secondly, few PCs were running truly mission-critical systems, so if the data became a little "damaged", who really cared? (Well, the companies involved cared very much indeed, but the software world wasn't too concerned.)

So the early PC RDBMSs passed responsibility for this level of control to the programmer. This meant that writing a totally secure database was perfectly possible in, say, dBase. The snag was that you had to be a good programmer and it took a great deal of effort. In addition, there was no centralised area where the relationships could be examined, so maintenance was difficult. If you suspected a join was being incorrectly supported, you had to hunt through, and understand all the relevant code, to find the area which was compromising the data.

As PC-based RDBMSs have grown up and come of age, there is now a strong need for a data dictionary. Sadly it has proved, shall we say, challenging to bolt a data dictionary onto those RDBMSs with a large installed base of users, code and data files. The result is that none of the classics (dBase, Paradox, FoxPro, etc) have acquired one, even with the major rewrites that these products underwent in moving to Windows. Many people (or is it just me?) feel that this was a major opportunity missed. The result for developers is that maintaining the integrity of the database remains their responsibility, and they probably have other things to worry about.

Some "modern" products, like Access, maintain a data dictionary and as a result do not inflict this extra workload on the developer. Less serious "modern" products, like Approach, which have chosen to use the dBase file format, have inherited the problems associated with that format.
9. Rules controlling data entry to specific fields must be storable in the data dictionary and applied at the table level.
For the same reasons given above, storing this kind of information centrally can protect data and reduce the developer's workload. Sadly, no RDBMS implements this as fully as it might. Most allow, say, input masks to be defined as part of the field definition so they are applied globally. However, as we have seen in this column over the last few months, input masks are often inadequate for controlling the input of real data such as telephone numbers and postcodes. Often, more sophisticated controls are needed. Most RDBMSs only allow these to be applied at the form level; it then remains the responsibility of the developer to make each form enforce the controls. The whole process leads to a waste of valuable (and hence expensive), human time and effort.

Tips & Tricks

Here are a couple of great tips from regular *Hands On* contributor Shane Devenshire, of Walnut Creek, California:
• "In Access, if you double-click the sizing handles on the right-hand side of a control, the control itself will 'best fit' its text. That is, it will change size to contain exactly the text in the control when you are in design view." *Nice one, Shane. At least in Access 2.0 it works with all of the resizing handles.*
• "You can also use the keyboard to move and resize controls. Move a selected control with ALT and the arrow (cursor) keys; to size the control, use CTRL instead of ALT.

"Surprisingly, even if you have Snap-to-Grid activated, these keyboard methods work without activating the Snap-to facility, making them great for fine-tuning."

Questions & Answers

Signalling problems
"I am writing a multi-user database application and want to be able to signal between two workstations, so that when one finishes a process, the other can start its work. How do I do this?"

Workstations on a server-based network (which I presume this to be) don't usually signal to each other directly, either in a database or any other applications. Instead they communicate with the server, so the best way to solve your problem is to use the server as the channel for communication. Create a table on the server called (for want of a better name) SIGNAL. This should have a single Yes/No field called FINISHED, and a single record, default value "no".

Cause the first workstation to write the value "Yes" into the field when it has finished, and get the second workstation to poll the value UNTIL it changes to "Yes".

You can make this more complex. For example, you could adapt it to create a new record every time the process changes hands, and store in the same table the time/date at which hand-over occurs. This pre-supposes that you actually want to know when the processes swap over, but this sort of information can be extremely useful.

Looking for lotto
"I am using MS Access for Windows. I have a 20-record file, each record having six numerical fields. I wish to do a search across all fields for six numbers. I know that it is possible to find which fields these are in individually, but is it possible to search across all records for a set of six numbers and find out which records they are in and how many are in each? As I have just acquired Access as part of Office, I have no experience of its functions or how it works. (For the record — a pun, *MW!* — this is associated with National Lottery numbers.)"
Mark Broadbent

Fig 1 (top) *The table LOTTERY stores your lottery guesses; in this case I have seven. Each week, into the table LOTTERY WINNER, you write the six winning numbers. Note that this table also stores the number of "Matches which are found between the winning number and your guesses"*
Fig 2 (above) *Given that this week's winning numbers are 1,2,3,4,5 and 6, I simply have to press the button labelled "Press Me!" and the number of matches appears*

I regard the National Lottery as a pernicious influence on the nation. However, this is an interesting database problem applicable to more than the lottery alone, and hence worthy of study.

Traditionally, RDBMSs assume that information will appear in a "known" field. Thus, if you want to find a customer called Smith, you search in CUSTOMERS.LASTNAME. However, the results of the National Lottery (about

Lottery: the guesses

Lot1	Lot2	Lot3	Lot4	Lot5	Lot6
1	2	3	4	5	47
1	2	3	4	5	45
2	3	45	46	47	49

Lottery: the answer

Lot1	Lot2	Lot3	Lot4	Lot5	Lot6
1	9	30	37	45	48

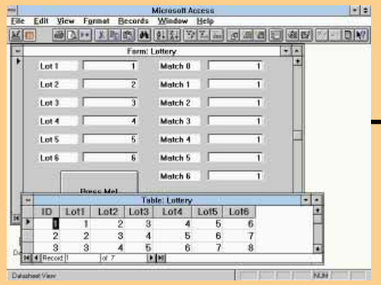


Fig 3 *It appears that I have won the lottery, because one of my guesses matches all of this week's numbers. Whoopeee!*

which I know very little) can, I believe, consist of six numbers. So, if you have three guesses, they might be as shown on page 279.

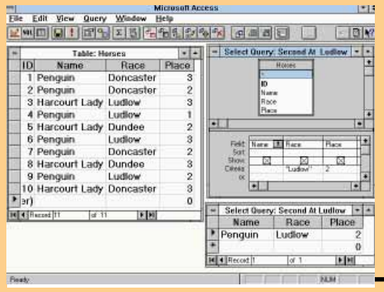
Now, given this week's answer (also page 279), I want to know how many matches each guess has attained. In the example given, two of my guesses contain the number one, as does the correct answer. This is easy to test for, since the number one always occurs in Lot1.

However, two of my guesses have the number 45, as does this week's answer. The trouble is that the number 45 is in Lot6 in one guess, Lot3 in another, and Lot5 in the actual answer. To calculate the number of numbers that match the answer, I have to compare each number in a given guess with every number in the answer. (Given certain sets of numbers, every comparison isn't necessary, but it is easier to write code which checks all combinations. This is less efficient, but deciding whether a comparison needs to be done takes more time than simply doing all possible comparisons).

As I said above, most RDBMSs assume that you know which field your data will be in — the problem here is that we don't. The only solution I could dream up involved code; can anyone find a solution using a query?

The complete application is on the cover disk as GAMB.MDB, including all of the code. You place your guesses into the table called LOTTERY and the winning answer into LOTTERY WINNER. Then you open the form LOTTERY, move to the record containing the correct winning number, and press the button. The number of matches appears on the right-hand side of the form (Figs 1 to 3).

Before you use it, a word of caution: I haven't tested this extensively, so if it tells you that you have won a fortune and then turns out to be lying, don't sue me — check it manually first. If you do win vast sums of money, please remember that I helped you to discover that you were a winner (hint, hint).



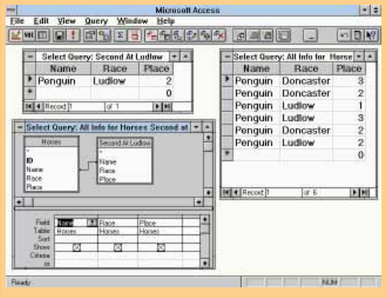


Fig 4 (left, top) *The table contains data about the horses and the races they have run. The query, top right, finds all of the horses which have ever come second at Ludlow. At bottom right you can see the result of that query: only one horse matches the criteria.* **Fig 5 (left)** *At top left is the query that you can see in Fig 4. At bottom left, a second query is using the results from the first one as its criteria*

Query 1 **Fig 6**

```
SELECT DISTINCTROW Name, Race, Place
FROM Horses
WHERE ((Race="Ludlow") AND (Place=2));
```

Query 2

```
SELECT DISTINCTROW Horses.Name, Horses.Race, Horses.Place
FROM [Second At Ludlow] INNER JOIN Horses ON [Second At Ludlow].Name =
Horses.Name;
```

Horses for courses

"Is it possible (using any mainstream database) to request the database to find the first record whose fields meet a particular set of criteria (say record X), search for the next occurrence of a record whose "name" field matches that of record X, then return to the first record following record X which again matches the required criteria?

"I've got a database of several years' horse-racing results and I want to be able to search for horses who display particular characteristics; then search forward in the database to see firstly whether they ran again that season and if so, whether or not they won, etc."

Iain Simpson

I've never received a question about gambling; and then two within a week. It's a perfect demonstration of the Theory of the Clumping of Rare Events.

Use two queries. The first finds the horses that match your criteria. For example, the HORSES table contains some minimal data about horses and racing. I can use a query to find the names of all the horses which have, for argument's sake, come second at Ludlow (Fig 4).

The second query again looks at the HORSES table, and it uses the names returned by the first query as the criteria for its search. So, if the first query returns the name "Penguin", the second query returns all of the data in the HORSES table which relates to the horse of that name (Fig 5).

You can set up multiple query pairs like these which look for different criteria. Once set up, all you need to do is run the second query in the pair (in this case the one called "All Info for Horses Second At Ludlow"). These sample queries and table are also in the same database on the cover disk.

For non-Access users, the SQL (Fig 6, alongside) may be useful.

PCW Contacts

Mark Whitehorn welcomes readers' correspondence and ideas for the Databases column. He's on **penguin@cix.compulink.co.uk**