Information emporia

Mark Whitehorn on the pros and cons of data warehousing. It's very trendy, but is it useful?

There is a great deal of talk nowadays about data warehouses. Some people are even building them.

I once carefully explained the principles of data warehousing to a friend. At the end of it, he said: "You mean, what I used to do ten years ago, without telling anyone, in order to make the system run faster?" The answer was "Yes". Very little is new in the database world; it just gets wrapped up in fancy words. On the other hand, the new words don't stop data warehousing being exciting, powerful and ultimately incredibly useful.

So, what is a data warehouse? We'll start from an operational database, defined in this case as a database into which one group of people regularly stuffs data and from which another group regularly extracts it. In other words, an operational database is a normal, regular kind of a database. But if you can put data in and get it out, why do you need a data warehouse? Well, because the first group of people generally ends up hating the second.

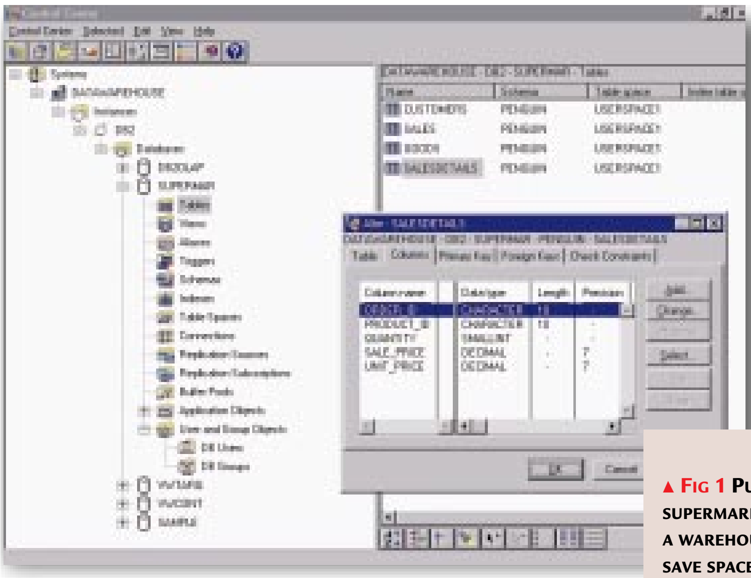
Questions, questions

The main reason for the resentment is that some queries can totally sock the performance of an operational database. Consider a sales database for a supermarket [Fig 1]. Given those iniquitous "loyalty" cards that are so popular nowadays, we can expect the CUSTOMER table to contain a reasonable level of detail. We can also expect the tables to rapidly become non-trivial in size (millions of records). When a questioner asks, "What is the value of all fish-related sales to childless couples in Bradford?" the operational system is totally thrown because so many records have to be hit to answer the question, even with decent indexing.

What is to be done? The answer is a data warehouse. At its most basic, this is simply a copy of the database held on another machine. The operational database continues to be used for data input but all queries are run against the data warehouse. True, depending upon when and how frequently the copy is made, the answers to the queries will be out of date, but queries of this nature are rarely that time-sensitive. And, once you have separated the input function from the querying, you can do really nice tricks.

For a start, you can de-normalise the data in the data warehouse. "Woah!" you cry. "This is the man who has been advocating normalisation for years in this self-same column!" Sure I have: it's very important in an operational database because it helps to ensure that data is correctly input. But once you know that a database will never be used for input, one of the major reasons for normalisation disappears. And, when you realise that your data warehouse is freed from this particular constraint, there are other changes you can make to the structure.

You might find, for example, that your "querying" users never need the fine level of detail inherent in an operational system. They are unlikely ever to want to know exactly what Fred Smith bought on 1/7/1998. So, you may be able to summarise the data as it is brought into the data warehouse. Clearly, at this stage, you need to talk carefully to the people who do the querying, to get a feel for what they typically run. However, the benefits can be immense, reducing query times from days to minutes.



Name	Schema	Table space
CUSTOMERS	PENGUIN	USERSPACE1
SALES	PENGUIN	USERSPACE1
BOOKS	PENGUIN	USERSPACE1
SALESDETAILS	PENGUIN	USERSPACE1

Column name	Datatype	Length	Precision
ORDER_ID	CHARACTER	10	-
PRODUCT_ID	CHARACTER	10	-
QUANTITY	SMALLINT	-	-
SALE_PRICE	DECIMAL	-	7
UNIT_PRICE	DECIMAL	-	7

▲ **Fig 1** Put your supermarket into a warehouse and save space

All becomes clear

You may now be thinking, "But if a data warehouse is simply a copy of an operational database, why are they so intimidating, expensive and complex? And why do they need such expensive software and hardware?" These are all excellent questions. The answers are, that given a small(ish) database, in a small-to medium-sized company, none of the above need apply.

A data warehouse is simply a way of working, an approach to manipulating data. It does not require huge complexity. There is nothing to stop you from setting up a query-only copy of your database simply to separate the query function from the input. This was exactly what my friend was describing.

Of course, once you start working with mission-critical, 24 x 7 x 365 systems, the data warehouse becomes complex. But then, the underlying operational database is already complex in these cases. For smaller systems, data warehousing can be a cheap, extremely cost-effective solution, and data warehouses offer other advantages — see next month's exciting episode.

TIPS AND TRICKS: COUNTING ON A RELATIONSHIP

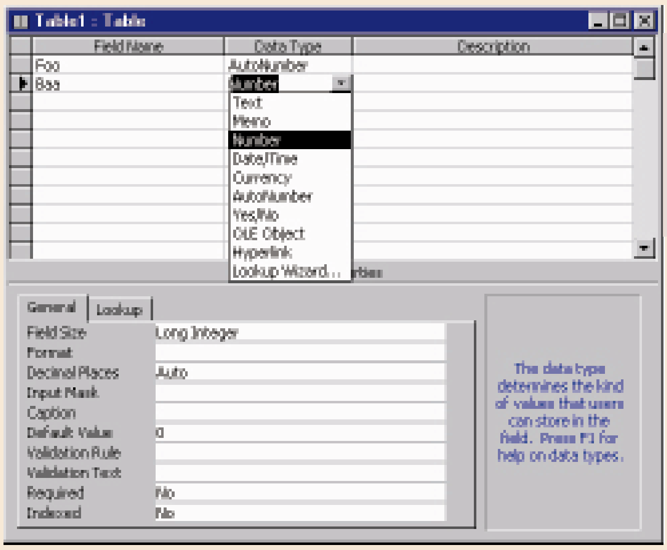


Table1 : Table

Field Name	Data Type	Description
Foo	AutoNumber	
Baa	Number	

Text
Memo
Number
Date/Time
Currency
AutoNumber
Yes/No
OLE Object
Hyperlink
Lookup Wizard...

Field Size	Long Integer
Format	
Decimal Places	Auto
Input Mask	
Caption	
Default Value	0
Validation Rule	
Validation Text	
Required	No
Indexed	No

The data type determines the kind of values that users can store in the field. Press F1 for help on data types.

◄ **The data type determines the kind of values that users can store in a field**

Here's an idea from Peter Thompson <atcthomp@mail.gcc.com.bh>. He says that if he uses a counter to set a "company ID" key field in a company table, and then tries to establish a relationship with enforced integrity with a "company ID" field in an order table, the action is refused because the fields are not identical (one is counter, the other is number). He overcame this in one of two ways. On the one hand, you could make a new field in the company table, number type, and write the counter to this field, by macro, when entering a new company. This can then be set as the key and you can make the relationship.

On the other hand, you could enter the first company as 1 in a numeric "company ID" field, then for each new company run a lookup of a query against company IDs to identify the largest number, add one, and write to the "company ID" field of the new company. Peter admits that both solutions are a bit untidy, but they solve the problem. He wonders: "Am I missing an easier way?"

Well, a brief bit of background first. Number fields in Access (and RDBMSs in general) can be sub-typed as Byte, Integer, Long Integer etc. The default sub-type for a numeric field in Access 2.0 is Integer. A Counter field is actually just a numeric field (sub-type Long Integer) which happens to have special incremental features. Therein lies the problem: a counter is a Long Integer and the default sub-type for a numeric field is Integer. These two are incompatible and hence you cannot join tables using these types.

I am impressed with Peter's workaround, but yes, there is an easier way. Simply set the numeric type to be sub-type Long Integer, and all should be well. This problem magically disappears in later versions because Auto-Number, the replacement for Counter, is still a Long Integer, but this is now the default for Numeric fields in general.

Recovery option

Reader John Blight <john.blight@triaster.co.uk> has commented on a recent *Hands On Databases* article (*PCW*, November '98) about recovering corrupted Access databases. He has found that the compact process is very valuable in correcting strange, often inexplicable behaviour in an Access database: the record printing from a form could well be such an example. If one of his users reports such a problem, his initial advice is usually to compact the mdb files. He doesn't know why the compact routine should be so effective in remedying corruptions, considering what it is designed to do, but it certainly appears to be so. This is in addition to making often considerable savings in file size, and more efficient operation.

John says that a useful facility is to have a compact (and/or repair) option under the right-click menu of mdb files:

1. In any Windows Explorer-type window, select Options... from the View menu, then the File Types tab on the dialogue box.
2. Select whatever file-type is associated with the mdb extension, then click on the Edit... button.
3. Click on the New... button.
4. In the Action text box, type "Compact an Access mdb" or something else suitable.
5. In the "Application used to perform action" text box, type in the code shown in Fig 2. "%1" is the mdb filename argument. Whether or not "%1" is within double quotes depends on whether the argument is being sent to an application which used long filenames; Access 2.0 doesn't, Access 7.0 and 8.0 do.
6. Click on OK.

When you next right-click on an mdb file, you should have the option to "Compact an Access mdb".

Nice one, John. I did know how to add to the right-click option, but had never thought to use it in this way. I've set up my own machine in the way you have described.

[FIG 2]

Type in:
```
<Path to Access 2.0> %1 /compact
```
or...
```
<Path to Access 7.0 or 8.0> "%1"/compact
```

PCW CONTACTS

Mark Whitehorn *welcomes readers' suggestions and feedback for the Databases column. He can be contacted via the PCW editorial office (address, p10) or email* database@pcw.co.uk.