In the frame

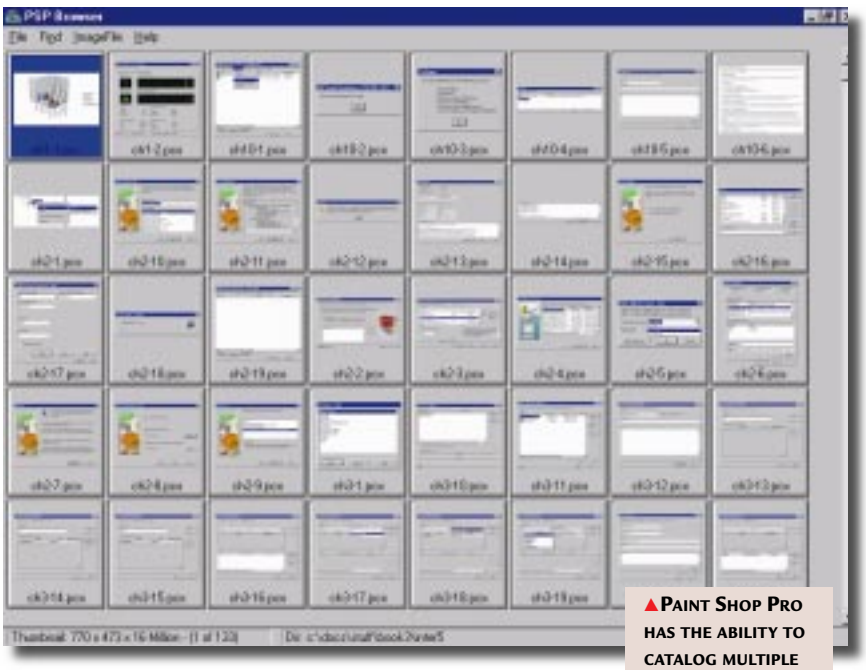
How do you store today's complex data structures in databases? Which comes first, the data or the table? Mark Whitehorn compares two views: UDB or connectivity?

You may be aware of rumblings in the database world about the storing of objects in databases: what objects are, how they should be stored and managed. It's becoming something of a quasi-religious issue, as is often the case when two strong contenders encourage polarisation.

What's it all about?

RDBMSs are good at holding data which has a relatively simple structure — in other words, data that fits neatly into a table. There's nothing wrong with that, but it is really the other way around: a table is a container that was designed to hold the sort of data (text, numbers, etc) which, back in the seventies, people, wanted to store and access. However, people are now trying to store more complex data which doesn't fit so neatly into the approved framework.

Consider a digital picture. On the face of it, there is no problem storing this in a table. After all, a digital image is just a long string of noughts and ones. You can think of it as being either a very long text string, or a very big, high-precision number. Either way, all you have to do is define a special data type to hold it. Indeed, many RDBMSs, including Access, have such a data type built in: the BLOB (Binary Large Object) data type. You can declare a field to be of this type and then pop your picture in there. No problem, except... this approach does nothing about giving the user of the database access to all the properties inherent in the original picture. In other words, a picture is more than a string of binary digits: it has one or more subjects, a dominant colour, a focal point, and so on. If it is an image of a painting, that painting probably has an artist, a title, a history, a meaning (...but let's leave modern art out of this discussion!). None of those object properties are stored in the BLOB and therefore the database cannot be queried on those properties.



▲ **Paint Shop Pro has the ability to catalog multiple images but is it an RDBMS as we know it?**

This is a shame, because human beings typically want to retrieve images on this sort of criteria, rather than asking for all the pictures that start with 011010111010111010. True, you can attach a viewer to the field that will enable users to look at the image, but that still doesn't help with finding it in the first place. Another approach is to attach one or more fields for keywords, to enable BLOBs to be located. But that adds manual graft and does not cover future searches which may need to scan for characteristics that, at present, we cannot even imagine.

RDBMSs are good at holding data that fits neatly in a table

Another example of problematical data is email. Much of my correspondence is now email so I store it for years because it documents my exchanges with other people — my own promises and commitments as well as those which others have made to me. An email is an object with a fairly well-defined set of properties associated with it: time/date of despatch and/or arrival, origin, whether it has been forwarded, its contents and so on. All my email sits in my email system which happens to be called Pegasus. I use Pegasus to search for emails by date, sender etc., whenever I want to locate an old email.

I love Pegasus dearly, but the truth is that it has rather poor search facilities. Why? Because it is not an RDBMS, it's a mail system. RDBMSs are optimised for searching: they have indexes and SQL to allow me to perform very devious searches. Pegasus doesn't; it is optimised to send and receive mail.

This brings us to the nub of the current debate, which is centred around the question of where such data (pictures, sounds, emails) presently stored in non-RDBMS sources, should be stored in future.

The UDB view

The UDB (Universal Database) argument is simple. It says that all data, no matter what, should be stored in a relational database for the simple reason that RDBMSs are optimised for storing and manipulating data. All we have to do is

to extend the relational model. The UDB supporters are fully aware that this means more than simply adding further complex data types. They know that the RDBMS will also have to incorporate viewers, complex "interpreters" which can "understand" the data contained in those data types. But this is being done as we speak. Systems already exist which cannot only store fingerprint images, they can also classify them so that comparisons can be made.

Having all data in RDBMSs earns some huge payoffs because it makes it so easy to correlate information from disparate sources. Currently, if I receive email from colleagues and want to ring them on the internal phone system, I have to swap to a web browser and look up their numbers on a web page. If my email system and the phone book were both back-ended by (say) Oracle, the system could automatically display the number each time I read an internal email.

The UDB approach is strongly supported by Oracle, IBM, Informix and Sybase, and all these companies produce some flavour of UDB. Now, just to prove that you're awake, hands up if you've guessed which company takes a contrary view. Let's do a hand count: 1... 2... Oh! All of you guessed.

The OLE DB view

Microsoft has no plans for a UDB offering. Instead, "the big M" argues that it is simply not realistic to expect all data to be stored in an RDBMS and that the typical, fairly crude data stores currently used by software like email packages will persist. So, rather than build a UDB into which you could put all the data but probably won't, Microsoft says let's keep RDBMSs more or less as they are and provide hooks from the RDBMS into other data storage mechanisms. The means of making these connections is the shiny new OLE DB from Microsoft (of course) which that company hopes will rapidly become as ubiquitous as ODBC.

▲AN (AS YET UNFINISHED) IMAGE BY MODERN ARTIST, STEPHEN BARCLAY. AUTOMATICALLY CLASSIFYING THIS IMAGE IN A MEANINGFUL WAY FOR HUMAN BEINGS IS A CHALLENGE THAT WILL TAX COMPUTER SYSTEMS FOR SOME TIME

Which one is right?

The really interesting question is, which approach is "correct"? The answer is that no-one knows; just like the JVC *versus* Betamax debate, this one won't be decided by technical competence but by the way the industry as a whole finally jumps. However, that has never stopped me from voicing my opinions in the past, so here goes.

As a database freak, I wholeheartedly support the UDB concept — no question. However, as a pragmatist I think that Microsoft is ultimately right. I believe that, realistically, there will always be data stored outside the RDBMS ideal, and that people will still want access to that data, so they will surely need OLE DB. Nevertheless, there is an extremely important rider to that viewpoint which strongly suggests that the story is not yet over. We are all aware that the registry in NT is less than perfect. It grows, it bloats, it acquires errors on machines where software is frequently loaded and unloaded. The result is that it is not uncommon for NT to be reloaded at six-monthly intervals on such machines, simply to clean up the registry.

Microsoft's argument skirts the issue that data stored by its own non-relational products brings problems by the barrowload

The fundamental problem is simple. The registry is not a proper transactional database — just like the email systems and all the other applications which store data in a loose and slipshod manner. In turn, this means that Microsoft's argument for the continuation of non-relational data stores skirts the issue that data stored by its own non-relational products brings problems by the barrowload. We live in interesting times.

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.