PASCAL

The blind leading the blind

I have long been deploring what I call the "cookbook" approach to becoming a database practitioner. There is hardly education to be had any more, including academia. It's been almost entirely replaced by product training, devoid of any history, and fundamental concepts and principles of the field. Those who undergo such training are not even aware that there are such things as concepts and principles beyond the product features they learn. They operate with blinders, because they essentially can only duplicate memorized recipes, and do not fully understand why various product features exist, what they *mean*, and whether they are correct or sufficient; when problems arise, they cannot associate them with the real causes, and cannot address them correctly without piling up additional problems.

Worse, some of these cookbook practitioners, who know even less than their trainers, end up training others, causing an accelerated dumbing down trend. The consequences are obvious.
Consider, for example, *A Short Oracle Tutorial For Beginners*, by a UK outfit called Smart-Soft:
This is just a quick introduction to Oracle for beginners, to give a short history of databases and Oracle Corporation's role in them, explain relational theory and provide a few examples so you can see how relational databases work. There is also a very brief discussion of object-oriented design as it applies to databases.
"Explain relational theory" and "object-oriented design as it applies to databases"? The latter applies to programming, not databases, and we'll shortly see about the former.
In the late 1960s/early 1970s, specialised data management software appeared – the first database management systems (DBMS). These early DBMSs were either hierarchical (tree) or network (CODASYL) databases – not relational or object-oriented – and were very complex and inflexible which made life difficult when it came to adding new applications or reorganising the data. The solution to this was relational databases which are based on the concept of normalisation – the separation of the logical and physical representation of data.

Relational databases are *not* "based on the concept of normalization". They are based on *predicate logic* and *set theory*. Normalization is a set of logical design principles for relational databases to avoid several problems: redundancy, update anomalies, complexity of queries and of interpretation of results.

The logical/physical separation is *not normalization*, but *physical data independence*. The article goes very briefly through some history of IBM's System R research and that of the Oracle Corporation and concludes:
As relational databases became accepted, companies wanted to expand their use to store images, spreadsheets, etc. which can't be described in 2-dimensional terms. This led to the Oracle database becoming an object-relational hybrid in version 8.0, i.e. a relational database with object extensions, enabling you to have the best of both worlds.

It's not relational databases that were accepted, but SQL DBMSs. As we have amply documented in our writings and seminars, SQL was the IBM research prototype language, which was thrown in the public domain without much thought. Because its authors did not have a proper grasp of the relational model (which is true to this day, see *If You Liked SQL, You'll Love Xquery*), it is very far from what a truly relational data language could and should have looked like. So even though almost everybody deems SQL DBMSs relational, they are really nothing of the sort. A sad consequence of the "cookbook approach" practiced in the industry.

We will probably forever have to reiterate again and again for the rest of our lives:

Relations are not two-dimensional; N-attribute relations are *N-dimensional*. Tables on paper or screen are *pictures* of relations, whose *medium* is two-dimensional, but they still have N columns and, therefore, represent N dimensions.

What is more, attributes can be of any type – text, images, audio, video, spreadsheets – you name it (see Chapter 1 in *Practical issues in Database Management*), so what Oracle should have done is implement the relational model with true user-defined data types, not SQL with object-extensions.

A relational database can be regarded as a set of 2-dimensional tables (known as "relations" in relational database theory). Each table has rows (known as a "tuples") and columns ("domains") and the relationships between the tables is defined by one table having a column with the same meaning (but not necessarily value) as a column in another table.
Tables are not "known as relations" in theory. Relations are *represented* by a special kind of table in databases. The

columns represent *attributes* whose values are drawn from data types. Only one kind of relationship in a relational database is represented by tables sharing columns defined over the same data type, and thus representing meaningfully comparable attributes. If the column in one table is a primary key, then the column in the other table is a foreign key.

> An object DBMS is nothing but a true RDBMS that supports type inheritance

Relational databases obtain their flexibility from being based on set theory (also known as relational calculus) which enables sets or relations to be combined in various ways:

- *via a join (also known as "intersect", or "and");*
- *union ("or", "add");*
- *exclusive "OR" (subtracted);*
- *and outer-join which is a combination of intersecting and exclusive or'ing.*

Relational calculus is *not* just another name for set theory. And relational operations are *not* just for combining relations.

Here's the "brief explanation" of object databases:
An object-oriented database, as the name suggests, stores and manages objects. In this context an object has both attributes and methods (a program stored within the object that performs a certain action or task) and in a true object-oriented database would belong to a class and would allow multilevel inheritance. The later versions of Oracle (Oracle 8, Oracle 8i and Oracle 9i) are object-relational hybrids because they support both relational and object-oriented features. The relational features are still the more prominent at the moment, but this will probably change as the industry begins to learn how to use the new technologies.

Got that? In fact, if correctly taken to its logical conclusion, an object DBMS is nothing but a true RDBMS that supports type inheritance, but correctly, not as it's implemented in current object DBMSs (see *The Third Manifesto*).

If you ever wondered why database practice is in decay, the proliferation of material such as the hereby debunked article is one reason why. This is the kind of introduction to database fundamentals – and the *only* one, if any – that practitioners get today, offered by people who have no clue. God save us all, and I'm an atheist!

For a proper introduction to fundamentals see the Practical Database Foundations series of papers, and the seminars for which they serve as text.

Fabian Pascal is onafhankelijk IT-analist, consultant en auteur gespecialiseerd in data management.