PASCAL

Assembling Cars, disassembled Minds

The tax authority demands that you (businesses) deliver several forms to them each year. You deliver them as electronic forms. Each form is defined as an object ... (sic, as you say). So, for each form, your name (and any other repeating information) is repeated physically ("encapsulated" in the "persistent" "object" ...). Denormalization at n'th degree (n = number of forms you deliver). The very fun part is that about the only information that the IRS wants, is your gross income: it is used to calculate your tax. Since that information can appear in two (or more) forms, which one contains the correct income?
(Personal communication)

A thread was initiated in a sqlteam.com forum with the following question:
Shailesh: To what extent one should normalize a given data model keeping in view performance for OLTP database? I have observed that most of ERP system databases are far denormalized, they maintain the data integrity procedurally and not through relationship constraints.
I guess the design/normalization of OLTP systems is such as to facilitate fast input of data into the database. And for such purpose denormalized versions would be just right (as implemented by ERP systems like JDEdwards).
One response was:
Brett: Well ... yeah ... they do that as in to be "flexible", or to be all things to all people ... PeopleSoft for example has no RI ... and then these Smarstream ... ugh EXTREMELY PAINFUL. Now, that's not to say denormalization doesn't have it's place ...
Now, without database integrity (of all four kinds, not just referential, see Conceptual Modelling and Database Design: A Foundation Framework for Data Management (http://www.dbdebunk.com/page/page/764907.htm, forthcoming at http://www.dbdebunk.com) things are, indeed, quite painful – that is, if you're aware of integrity at all, which is increasingly rare. But 'flexibility' is a rather poor choice of words, because one of the purposes of normalization is more flexible databases, both for concurrent users with different data view needs, as well as for changing such needs over time. Besides, it begs the question:
Jay White: Curious ... what is it's place?
Which got the following response:
X002548: I denormalize for OLAP, not OLTP.. I had a bunch of web developers that wanted to dynamic against a 2 dimensional hierarchic table (Org x Level of service) ... I protested ... management sided with the "flash" of RAD ... And when the pages got served slower than a french restaurant, they decided that it might be in their interest to build an overnight batch process to speed up the process ... Now where did I hear that before ... Of course it was all their idea in the end ...
The table in question is not one that represents a set of entities in the database, but rather an analytical one, a specific data view, more akin to a query result, or report. It is the job of an analytical application to operate on properly designed relational tables to produce such a result. To the extent that performance is poor, the reason is not normalization, but the physical implementation details of the DBMS and database, and the quality of the application (efficient implementation of truly relational DBMSs should do just fine, see More on Final NULL in the Coffin (http://www.dbdebunk.com/page/page/1649301.htm); and as it turned out, a denormalized solution did not exactly yield exciting performance either, see below for more on OLAP).
Page47: Well, for fun I went out and did some searching for denormalization for performance ... And then I read some stuff by Fabian Pascal on the concept. While certainly interesting, Fabian's arguments really have little application to SQL based programmers and designers. He states this pretty clearly in his articles, mentioning several times that SQL DBMS are not truly relational, and do not separate the logical table structure from the physical table structure well enough. I do realize that all of his arguments are not invalid, and that his advice is valuable and knowledgeable, but it is also true that denormalizing the physical table structure of a SQL database does lead to application specific improvements. Given that I haven't heard of too many file-based applications out there that match the performance and flexibility of a SQL DBMS, I don't see why, in some cases, it isn't permissible to use a SQL DBMS in a somewhat denormalized form to accomplish what you are hoping to do. Should it be used willy-nilly by people who don't really understand it well? Certainly not. Should you make every effort to increase performance in other ways first? Absolutely. But when it comes down to it, sometimes it's necessary.
Now, it is indeed, correct, that direct-image SQL DBMS implementations provide weak support of data independence, and their sets of physical structures and access methods are not as rich as they could and should have been; and that, as a result, performance is sometimes-thought by no means, always – poor when databases are normalized. But even that does not mean that denormalization is the solution. As I demonstrated in The Costly Illusion: Normalization, Integrity, and Performance (http://www.dbdebunk.com/page/page/1103793.htm), gains in performance, if any, are not due to denormalization, but to failure to implement the additional integrity constraints to prevent possible inconsistencies due to the introduction of redundancy in the database:

Jay White: Just be sure that when you are capturing performance metrics against a denormalized schema, you take into consideration the added costs of maintaining data integrity. With that in mind, if the denormalized schema still comes out on top ... so be it. In my experience, most of the time, by the time I add in all the DML to maintain data integrity for a denormalized schema, I find that I was better off normalizing. YMMV.

The fact is that most practitioners are unaware of the need for such constraints altogether, and essentially trade off integrity for performance; besides, SQL DBMSs do not support the full set of integrity constraints necessary for relational databases, so even if they wanted to deploy those constraints, they couldn't. And because figuring out and declaring such constraints is prone to error and prohibitive, it makes no sense to bother with them just to get the same performance as with normalized databases, which makes them unnecessary.

If performance is poor, blame the real culprit-implementation – and stop accepting inferior products and integrity risks.

But what about "read-only", or rarely updated databases?

crazyjoe: But if absolute data integrity isn't necessary on the denormalized fields, you can manage without all of the extra constraints. Beyond that, a lot of DW [data warehouse] applications have a certain cyclical process ("month end") where the DW is unavailable for a couple of days while all of the ETL is done and then it's brought back up for users again. It's a large loss of elapsed time, but generally acceptable in the DW arena to maintain decent report performance throughout the rest of the month.

The notion that OLAP/DW data do not require integrity is somewhat of an illusion. DWs must be populated quite often and, given that most OLTP databases that are denormalized do not deploy the necessary constraints, chances are that their integrity is questionable. In absence of integrity enforcement at the DW level, the risk of inconsistent data is considerable (note that the same is true of even historical data, which is also read-only).

The point is that correct implementations of truly relational DBMSs (TRDBMS) would make obviate the need to accept such time losses and integrity risks. But as long as practitioners, pundits, and the trade press erroneously believe that denormalization, rather than physical implementations, is the problem, and accept this state of affairs, there is no reason for better products to materialize.

Perhaps the finest example in the thread of the state of knowledge in the industry is the following:

bm1000: There used to be a saying about storing data in a normalized database: It makes as much sense as getting up in the morning, assembling your car, and driving to work. Then, at the end of the day, driving home and disassembling your car into its component pieces. The moral of the story was to store data the way you use it. In my opinion, normalization is a process that you go through when developing a logical data model. Once the logical data model is stable, you then develop the physical database design. The physical database design is based on the logical data model, the processes that you are going to automate, and the service level objective that the client has agreed to. If you can achieve the performance requirements of the service level objective without denormalizing, terrific. But if you are having problems meeting your SLO, denormalization may well be the answer.

This "disassembling car" fallacy is an old one, regurgitated over and over, despite the fact that it has been debunked more than once e.g. On "Respected Technical Analysts" (http://www.dbdebunk.com/page/page/622709.htm).

One reaction got it almost right:

derrickleggett: I feel a warm, fuzzy feeling inside after reading this big line of crap. I'll have job security for many more years, going behind people fixing all the screwups because they were more concerned about the here and now, and totally missed the boat on the big picture. Ahhhhhh, the life of a DBA!!!! How many of you guys are DBAs? Normalization is building a car with standardized parts, instead of just "piecing it together" from whatever is available. You then don't have to spend years searching for a compatible part when something breaks instead of tearing the whole car apart and starting from scratch.

But not quite. A more suitable metaphor would be to some sort of vehicle used by multiple users for multiple purposes, say, carrying heavy loads, racing, and city driving, at different times; purposes which would also themselves change over time. So to which of the "how you use it" specifications would the vehicle be built? If such an animal were to be, it would have to be modular, and assembled and disassembled for each different use. If it does not exist, it is precisely because what is necessary and possible for computer applications, is not so for cars.

Moreover, "physical model" meant storage and access methods, we would agree with the separation and insulation of the logical of the logical from the physical design. We have reasons to suspect, though, like so often is the case, the two levels are confused: SQL base tables are physically stored as such and the direct-image implementation facilitates such confusion and, therefore, the notion that denormalization is the solution.

We cannot but concur with some reactions in the thread, but those are becoming increasingly scarce:

Michael Valentine Jones: Usually the people talking about de-normalization like this don't really understand normalization, so it is just BS for "I don't really know how to normalize a database, so I just threw some crap together, and said I did it for performance reasons". What I think is funny is that the no-nothings are the ones building the applications, and then the senior people are the ones brought in to fix the mess. Kind of like having the construction laborers build a skyscraper, and then bringing in the Architects and Engineers to figure out how to keep it from falling down.

AndyB13: Or putting the LUNATICS in charge of the ASYLUM!!

Fabian Pascal

Fabian Pascal is onafhankelijk IT-analist, consultant en auteur gespecialiseerd in data management. Zie ook zijn website www.dbdebunk.com