

Re: Business objects, subset of collection

Source: <http://coding.derkeiler.com/Archive/General/comp.object/2008-01/msg00280.html>

- *From:* "H. S. Lahman" <hsl@xxxxxxxxxxxxxxxxxxxxx>
 - *Date:* Mon, 21 Jan 2008 22:52:48 GMT
-

Responding to Frebe...

I solve particular problems using "the RDB query model"
every day, and
I have not noticed it being "very inefficient".

That's because you are either doing CRUD/USER applications or you
haven't benchmarked your solutions...

Once again, is the statemet below CRUD?

```
select i.invoiceid
from invoice i join payment p on i.invoiceid=p.invoiceid
group by i.invoiceid
having sum(p.amount) < i.amount and datediff(now(), i.duedate) >= 10
```

This is the kind of code I write every. Even though the number of
invoices and payments are very high, the queries perform within a
number of millis. The customer is happy, I am happy.

Once again, if that is all you are doing, that is CRUD/USER processing; you are just moving piles of data
back and forth between the UI and the RDB.

Maybe you can show your code performing the same task.

You want code for inventory forecasts? A Linear programs to allocate advertising budget to various markets'
media? A simulation model of atmospheric diffusion? You are basically asking me to show you code for an
entire application that solves some other problem that view conversion between UI and RDB. I don't think so.
[Note that the smallest application I have worked on in the past four decades, inside IT and out, was 250
KLOC. Non-CRUD/USER applications are not the sort of thing one tucks into eMails.]

That paradigm is fine for generic data

Re: Business objects, subset of collection

storage and
access but searching large sets sucks for
algorithmic processing.

SQL databases sucks for searching large data sets, come on...

You don't deny my assertion that I can perform the same $O(\log N)$
optimization in the implementation of a <reusable> collection class.

Of course not. The difference is that you have to do it by yourself. I
can reuse existing tools instead. That is the main difference between
using a database and not using a database.

And, as I have said at least twice, the price one pays for superior performance is hand crafting of the
optimization. In CRUD/USER processing that is largely irrelevant because (a) data is accessed once by each
solution and (b) seek time dominates the access performance issues in those situations.

But
if OO relationships are instantiated at the object (tuple) level, then
the N in OO searches will usually be much smaller than the N in table
level searches.

Since you qualify with "usually", this statement is pretty pointless.

I don't agree. The 'usually' applies in the relatively rare situations where the requirements specifically require
class/table level searches. Unless the application is just a pipeline between UI and RDB, that situation rarely
arises.

So one /must/ be able to achieve more efficient searches
given OO's object-level instantiation. The price one pays for that
efficiency is that the object-level instantiation has to be hand-crafted
based on the particular problem context.

Yes, I agree that by writing the corresponding code by your self, you
might get faster applications than you would if you had used a SQL
database.

But "usually" development time, cost more than CPU time. And the only
thing you have showed is that databases has complexity $O(\log A)$ and a
tailored solution $O(\log B)$, there $B < A$. This is not a very big deal,
compared to the "extra tons of keystrokes", you would have to do.

Re: Business objects, subset of collection

It /is/ a big deal when the same data aggregate is accessed many times. Besides, that was just one example chosen because the implications were so obvious.

Why do you think that designers of large, complex applications spend so much effort trying to minimize persistence access, regardless of whether they are doing OO or not? They introduce elaborate caching schemes, deliberately present themselves with major data integrity problems, and whatnot because persistence access is almost always the performance bottleneck of such applications.

Does

"insert into payment (invoiceid, date, amount) values (5, '2008-01-19', 500)"
has anything to do with persistence?

Caching is managed by the database. Caching is not the concern for the application designer.

Nonsense. Every non-CRUD/USER application I have ever seen had a client-side cache in some form. By the time the requests get to the DBMS, it is too late. The DBMS caches just manage server access and resources (unless you are talking about memory mapped OODBs). One caches on the client side to minimize DB access.

BTW, note that in an OO application as soon as one instantiates objects in memory using attributes values from the DB, one is providing a sort of client-side cache.

You are absolutely right that your solution introduces major data integrity problem. That is what I have been trying to show. That is why we should use a database, if possible.

Persistence is probably the bottleneck for many kind of applications. That is why databases using caching. In a well-tuned database 98% of all calls, doesn't cause disk access.

Persistence is /always/ the bottleneck when the solution needs to access the same data repeatedly.

When solving complex problems the same data is quite often accessed and processed many times during the course of the solution.

Is this the criteria for a "complex problem"?

Re: Business objects, subset of collection

It is one criteria. It is a sufficient condition, though not a necessary condition.

If one used the same accessing paradigm internally in the solution as the RDB uses the application would brought to its knees.

There are scenarios, there mainstream SQL databases wouldn't perform well. But they do excel in many application areas. The major problem with current SQL databases, is the limited set of index types that is used. B-trees is used as a one-size-fits-all solution. Obviously better support for other index types is wanted. I can imagine scenarios there foreign keys implemented as pointers would be a good thing too. As a matter of fact, I think such databases exists, even if I don't have time to find references to support this claim.

SQL databases excel in CRUD/USER processing, which is what I keep saying. Conversely, they suck once one gets out of that realm.

As an example, I play an MMORPG. The game employs an RDB, a star client/server model, and thin clients. That is a great architecture for airline reservations systems but it makes the MMORPG non-scalable. That's because the same data is constantly being accessed and updated by groups of clients involved in the same interactions (i.e., many times per second). The game provider is throwing hardware at the problem and doing handstands to balance loads but the lag just keeps getting worse as the number of players increased from 5K to 30K. I figure the game will simply die when the player count hits 50K because it will be unplayable.

So the cardinal rule of complex application development is to read the data once and write it once, no matter how many times it must be accessed in the solution.

Read and write once from what, disk? Or RAM?

The DBMS. Disk seek is the big problem but the table-level searches are still an important problem when the same data is accessed repeatedly in a single problem solution. (Or there are many possible relationships among the data or the data is constantly being updated.) It is also important to be able to convert identity into formats where one can use more efficient data structures, like arrays. (Note that does not necessarily mean pointers; consecutive integers work well too.)

But the 'n'
in
 $O(\log n)$
will usually

Re: Business objects, subset of collection

be much
smaller in
the OO
application
because the
collections
are
object-based
rather than
class-based.

Lets say you want to find all
unpaid invoices. Why would
the n be much
smaller in a OO solution?

I said, "usually". You are postulating a
class-based search as a problem
requirement.

You might think that my example is too extreme, but isn't it
good to
use a method/tools that doesn't limit you to work on small
amounts of
data?

Who is limited to working on small amounts of data?!?

You complained about my example, since it was a class-based search,
and not a search which could be solved with a limited number of
objects in a collection. So why don't show how you would find all
unpaid invoices?

I've already point out that you are postulating a requirement that /requires/ a table level search. [I have also
pointed out that even in that situation there are possible ways to avoid it, such as a collection dedicated to just
unpaid invoices.]

You are playing Topmind forensic games here: making a patently false assertion and then deflecting when
called on it.

Pointers are a red herring. They just provide a more efficient mechanism
for object (tuple) identity than embedded identifiers.

This in generally not true for data mangement. Pointer-based databases
hardly exists any more. Modern databases uses emedded identifiers.

Re: Business objects, subset of collection

So what? As I have said repeatedly, RDBs provide generic data access and embedded identifiers are quite convenient for that since identity has been cast to data. The issue is solving specific problems efficiently in a computational environment where one has address manipulation, structures like arrays, and whatnot.

My main problem is with table-based searches. Table-based searches are a very general access mechanism that provides uniform access regardless of usage context (exactly what one wants in enterprise data storage).

You seem to continue to claim, that SQL database doesn't perform well enough. I think it is up to the reader to judge, if this is true or not, considering all existing applications using SQL in many different areas.

That is not what I claim at all. Quite the contrary, I have said several times that RDBs provide an excellent paradigm for generic data storage and access. What I claim is that when solving specific problems in a computational environment like a computer there are more efficient paradigms for /manipulating/ data.

[The WHERE clause I used in an example is actually very rarely seen in OO applications; it is a kind of last resort when the requirements allow no more efficient alternative. And using a WHERE is very likely to get close scrutiny by OOA/D reviewers.]

Since WHERE is not supported by OOPLs, this is obviously true.

The first clause is not true. The OOPLs provide several mechanisms for implementing AAL WHERE clauses. If they didn't it would not be possible to provide a code generator for AAL.

Are producing invoices, "complex processing"? There are scenarios of data processing that are not supported very well by existing index types in current mainstream (SQL) databases. But the major part of all processing done in common business applications, perform very well using SQL databases, without having any custom data structures. What about all COBOL applications out there with extremely simple data

Re: Business objects, subset of collection

structures, relying heavily on SQL statements to do the job.
Doesn't
they perform well enough?

In most situations producing invoices is pretty straight forward. Order
+ Lading Bill = Invoice. IOW, I would normally expect that to be
CRUD/USER processing. That's why companies buy OTS Accounts
Receivables
packages rather than reinventing the wheel.

I could continue to give you examples of common tasks from business
applications, and you would classify them as CRUD as well. I think it
is honest by you to make your disclaimer about OO solutions not being
appropriate in all scenarios. You are almost the only one at
comp.object, doing so. The problem is that knowing when something is
CRUD or not. You claim that 20% of business applications out there are
CRUD, I claim 80%.

The point is that one can construct an Invoice entry from an Order entry and a Lading Bill entry by simply
moving field values from one view to another. That sort of view conversion is quintessential CRUD/USER
processing. That sort of view conversion was ubiquitous in core accounting packages that were the bulk of IT
through the '60s.

But those problems have all been solved and one can buy the solution OTS rather than reinventing the
solutions. So most CRUD/USER processing today just supports ad hoc desktop analysis. The important IT
problems today look a lot more like R-T/E or scientific programming as they struggle to deal with
concurrency, cutting edge data warehousing algorithms, networking, catalog navigation, highly complex
operations research algorithms, and a bunch of other problems that were not even feasible to solve in the '60s.

As I indicated, 20% was a guess, but the proportion is certainly a whole lot smaller than it was in the '60s.

My databases solves problems for my applications. Between
applications, data are exported and imported.

So you live with the reuse problems and the need to keep revising
schemas as new applications come online.

The last decade, I have been working as an independent software provider.
Usually I don't allow other applications to access databases for my
applications. Most software companies like SAP etc, doesn't allow
other applications to access their database either.

Good work if you can get it. But where do you get the data in your database? From user data entry? Seems a

Re: Business objects, subset of collection

pity existing data can't be used.

When one creates a Data Model, it should be based on the overall problem domain rather than specific problems within the domain. If you do that, then the same schema should be reusable by all the core accounting applications.

One could also say that it only need to be one core accounting application. The question is: How do you define the "overall problem", and the "specific problem". You base almost all your argumentation on such fuzzy terms, which makes everything derived from it rather fuzzy too.

I didn't say overall problem; my use of 'problem' was as an adjective. Problem domains are quite different things from problems. The point was that Data models should be constructed based on how the overall business environment works, not how individual problems are solved.

The words "overall" and "individual" are such fuzzy in this context, that any argument based on them are rather pointless.

Speaking of being pointless, we have been down this road several times before without resolution and I see no indication of it this time. We are just repeating ourselves. I think it is time to break it off.

—
There is nothing wrong with me that could not be cured by a capful of Drano.

H. S. Lahman
hsl@xxxxxxxxxxxxxxxxxxxxx
Pathfinder Solutions
<http://www.pathfindermda.com>
blog: <http://pathfinderpeople.blogs.com/hslahman>
"Model-Based Translation: The Next Step in Agile Development". Email
info@xxxxxxxxxxxxxxxxxxxxx for your copy.
Pathfinder is hiring: http://www.pathfindermda.com/about_us/careers_pos3.php.
(888)OOA-PATH

Re: Business objects, subset of collection