

Re: Business objects, subset of collection

Source: <http://coding.derkeiler.com/Archive/General/comp.object/2008-01/msg00261.html>

- *From:* frebe <frebe73@xxxxxxxxxx>
 - *Date:* Fri, 18 Jan 2008 21:48:06 -0800 (PST)
-

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.

Maybe you can show your code performing the same task.

That paradigm is fine for generic data 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.

Re: Business objects, subset of collection

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.

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.

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 persistance?

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

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.

Re: Business objects, subset of collection

Re: Business objects, subset of collection

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"?

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 where foreign keys implemented as pointers would be a good thing too. As a matter of fact, I think such databases exist, even if I don't have time to find references to support this claim.

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?

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

Re: Business objects, subset of collection

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?

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

This is generally not true for data management. Pointer-based databases hardly exist anymore. Modern databases use embedded identifiers.

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.

[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.

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

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

Re: Business objects, subset of collection

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%.

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 indepent 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.

If you allow access from other applications, you usually create tailored view for the access.

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 enviroment works, not how individual problems are solved.

Re: Business objects, subset of collection

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

//frebe

.