The dangers of hyper-normalization?

Asked 15 years, 11 months ago Modified 15 years, 11 months ago

Viewed 3k times

3

Several colleagues and I are faced with an architectural decision that has serious performance implications: our product includes a UI-driven schema builder that lets non-programmers build their own data types for a web app. Currently, it builds properly normalized schemas behind the scenes and includes some complex logic to alter the schema and migrate legacy data automagically if the admins make changes to the data types.

The normalized schemas have hit performance bottlenecks in the past, and a major refactoring has been scheduled. One of the groups of developers wants to store every property of the data types in a separate table, so that changes to the data types will never require schema altering. (A single property could be turned into a 1:n relationship, for example, just by changing application logic.)

Because early benchmarking indicates this will exact a huge performance penalty, they have built a caching layer in application code that maintains denormalized versions of each data type. While it does speed up the queries, I'm skeptical about the complexity that the application layer will be taking on, but I'm hoping for feedback - am I being pessimistic? Have others deployed this type of solution

successfully? Should I stick to my guns, or is moving the complexity from "Schema modification tools" to "Schema mirroring tools" a good thing?

sql database-design architecture

Share

Improve this question

Follow

asked Jan 20, 2009 at 6:31

Eaton

7,415 ● 2 ● 28 ● 24

4 Answers

Sorted by: Highest score (default)

11

> The normalized schemas have hit performance bottlenecks in the past, and a major refactoring has been scheduled. One of the groups of developers wants to store every property of the data types in a separate table, so that changes to the data types will never require schema altering. (A single property could be turned into a 1:n relationship, for example, just by changing application logic.)

This sounds like a bad idea to me.

1. It's going to mess up your database performance. If you store these things on one row they will be physically located together on the disk and treated as one thing for the purposes of locking, etc.

2. Queries that you write are going to require a mass of extra joins and will be very painful. You will end up writing views to turn it back into what it should have been in the first place.
3. The scenario described might never happen so you slowed down and complicated your application for potentially no benefit,
4. If it does happen and you are going to have to re-code and test a bunch of application code what's the little extra effort in making a database change at that time? You can make your child table, copy the data down into it with an update, and drop teh column from the parent table
5. If you are successful, in the future a different application may attach to your database. They will not be able to tell what the real schema is, because that information is held by your application. Model data in the database.
6. The cache on your application server can get tricky if (a) there is too much to fit in memory, (b) you scale to multiple application servers, (c) you have a different application that connects to your database. You're working around a performance problem that is of your own making.
7. You are not going to be able to create an index on multiple columns if they are each living in a child table.

 edited Jan 20, 2009 at 7:03

Follow

answered Jan 20, 2009 at 6:48

WW.

24.3k ● 15 ● 97 ● 124

All very good points and well explained. Wish I could upvote twice – Otherside Jan 20, 2009 at 7:13

If I hit him on the conceptual side, and you nail him on the logical side, maybe we'll derail this thing. :D – dkretz Jan 20, 2009 at 7:15

Well, so far the answers are confirming my own skepticism about this proposed change. Thanks for the clarifying arguments... – Eaton Jan 20, 2009 at 7:23

5

What you describe doesn't resemble what I call normalization. It's more like hyperabstraction - trying to find some abstraction level from which everything else can be derived. Like "Object" in javascript. If you take it to its logical conclusion, you could get by with two tables; one table for every Object, with a column for a ObjectTypeCode and ObjectId; and another table with Associations, having two ObjectId columns, a third for uniqueness, and a fourth for Value.

I suggest you need to revisit your domain model. The one you describe sounds scary (but unfortunately, eerily familiar) to me. I had a guy who worked for me who invented a table called "Objects". There were two child tables, ObjectAttributes and ObjectProperties. It was

difficult to get a clear explanation of the difference between the two. The table (and he, fortunately) didn't last long.

Share Improve this answer Follow

edited Jan 20, 2009 at 7:18

answered Jan 20, 2009 at 6:55

dkretz

37.6k ● 13 ● 83 ● 140

2

Strictly speaking, there is no such thing as "hyper normalization". If a fully normalized schema is refactored into another equivalent fully normalized schema, they are both equally normalized. What is often called "hyper normalization" is actually table decomposition in pursuit of goals other than normalization.

If I read you right, I don't think that's what's going on in the situation you outlined in your original post. It seems as though your group is debating between a normalized schema and a somewhat denormalized schema.

When this is done manually by an intelligent designer, there are trade offs between the performance advantages obtained with certain kinds of denormalized designs, and the update problems that follow, willy-nilly, from denormalization. I expect that your product that does the same thing automatically suffers from the same difficulties

that intelligent human schema designers suffer from, and then some.

I'm guessing that the real problem at your site is not really a performance problem at all. Rather, I think it's the conflict between the requirement to let users define new data types at will and the desire to obtain the same results one gets from a well designed database that's derived from a static set of information requirements.

These two goals are, in my opinion, irreconcilable. The ultimate layer of data management in your database is being done by your community of users. And they probably don't even meet in committee to decide which new definitions will become standard and which ones will be discarded. Unless I miss my guess, the schema is probably full of synonymous data types, where different users invented the same data type, but gave it a different name. It probably also contains some homonymous data types, where different users invented different data types, but gave them the same name.

Every time this has been attempted, it has resulted in chaos. The first visible sign of that chaos is an intractable performance problem, but that's just the tip of the iceberg. Sooner or later, you are going to discover that what you have in your database is unmanaged data. And you are going to discover that it's impossible to deliver the same bang for the buck that one obtains with managed data.

If I'm wrong about this, I'd love to learn about the example that proves me wrong. It would represent a fundamental

advance in the state of the art.

Share Improve this answer

Follow

answered Jan 20, 2009 at 14:38

Walter Mitty

18.9k ● 2 ● 31 ● 59

This is an excellent statement of the problem, ie. a conflict between the requirement for high abstraction and the desire for performance. However, friendly criticism: it's not an answer :-) – Jared Beck May 4, 2011 at 5:59

0

Our gracious SO team have tackled this subject: Maybe Normalizing Isn't Normal.

Conclusion -

> **As the old adage goes, normalize until it hurts, denormalize until it works**.

Share Improve this answer

Follow

answered Jan 20, 2009 at 7:32

gimel

86.1k ● 10 ● 79 ● 104