Scaling Secret #2: Denormalizing Your Way to Speed and Profit

Thursday, August 16, 2007 at 11:16AM

Todd Hoff in Shard, Strategy

Alan Watts once observed how after we accepted Descartes' separation of the mind and body we've been trying to smash them back together again ever since when really they were never separate to begin with.

The database normalization-denormalization dualism has the same mobius shaped reverberations as Descartes' error. We separate data into a million jagged little pieces and then spend all our time stooping over, picking them and up, and joining them back together again.

Normalization has been standard practice now for decades. But times are changing. Many mega-website architects are concluding Watts was right: the data was never separate to begin with. And even more radical, we may even need to store multiple copies of data.

Information Sources

- Normalization Is for Sissies by Pat Helland
- Data normalization, is it really that good? by Arnon Rotem-Gal-Oz
- When Not to Normalize your SQL Database by Dare Obasanjo
- MegaData by Joe Gregorio
- Audio of talk by Adam Bosworth at the MySQL Users Conference 2005

We normalize data to prevent anomalies. Anomalies are bad things like forgetting to update someone's address in an all the places its been stored when they move. This anomaly happens because the address has been duplicated. So to prevent the anomaly we don't duplicate data. We split everything up so it is stored once and exactly once. Bad things are far less likely to happen if we follow this strategy. And that's a good thing.

The process of getting rid of all potential bad things is called normalization and we have a bunch of rules to follow to normalize our data. The price of normalization is that when we want a person's address we have to go find the person and their address in separate operations and bring the data together again. This is called a join.

The problem is joins are relatively slow, especially over very large data sets, and if they are slow your website is slow. It takes a long time to get all those separate bits of information off disk and put them all together again. Flickr decided to denormalize because it took 13 Selects to each Insert, Delete or Update.

If you say your database is the bottleneck then the finger is pointed back and you and you are asked what you are doing wrong. Have you created proper indexes? Is your schema design good? Is your database efficient? Are you tuning your queries? Have you cached in the database? Have you used views? Have you cached complicated queries in memcached? Can you get more parallel IO out of your database?

And all these are valid and good questions. For your typical transactional database these would be your normal paths of attack. But we aren't talking about your normal database. We are talking about web scale services that have to process loads higher than any database can scale to. At some point you need a different approach.

Many mega-scale websites with billions of records, petabytes of data, many thousands of simultaneous users, and millions of queries a day are doing is using a sharding scheme and some are even advocating denormalization as the best strategy for architecting the data tier.

We sees this with Ebay who moved all significant functionality out of the database and into applications. Flickr shards and replicates their data to reach high performance levels. For Flickr this moves transaction logic back into their application layer, but the win is higher scalability.

Joe Gregorio has identified some common themes across these new mega-data systems:

- Distributed - The data has to be distributed across multiple machines.
- Joinless - No joins, and no referential integrity, at least at the data store level.
- De-Normalized - De-normalization is needed if you are avoiding joins.
- Transcationless - No transactions

It's the web model pushed to the data tier. Ironically, it may take a web model on the back-end to support a web model on the front-end.

The Great Data Ownership Wars: The Database vs. The Application

A not so subtle clue as to who won the data wars is to look at the words used. Data that are split up are considered "normal." Those who keep their data whole are considered "de-normal." All right, that's not what those words mean, but it was to good to pass up. :-)

Traditionally the database owns the data. Referential integrity, triggers, stored procedures, and everything else that keeps the data safe and whole is in the database. Applications are prevented from screwing up the data.

And this makes sense until you scale. Centralizing all behavior in the database won't mega-scale as the web does, which is why Ebay went completely the other way.

Ebay maintains data integrity through a service layer that encapsulates all data access. The service layer handles referential integrity, managing replicated copies, doing joins, and so on. It's more error prone than having the database do all this work, but you are able to do scale past what even the highest end databases can handle.

All this sharding and denormalization and duplicating at one levels feels so wrong because it's so different than we were all taught. And unless you are a really large website you probably don't need to worry about this level of complexity. But it's a really fascinating and unexpected evolution in design. Scaling to handle the world wide web requires techniques and strategies that are often at odds with our years of experience. It will be fun to see where it all leads.

Related Articles

- Flickr both denormalizes and duplicates data. Horror!
- Ebay is the most radical in moving almost all functionality out of the database and into the application.
- Plenty of Fish also advocates denormalization as a key strategy.
- Hadoop - a framework for running applications on large clusters of commodity hardware using a computational paradigm named map/reduce.

Article originally appeared on High Scalability (http://highscalability.com/).