

Chapter 1: Background

Introduced by Michael Stonebraker

Selected Readings:

Joseph M. Hellerstein and Michael Stonebraker. What Goes Around Comes Around. *Readings in Database Systems*, 4th Edition (2005).

Joseph M. Hellerstein, Michael Stonebraker, James Hamilton. Architecture of a Database System. *Foundations and Trends in Databases*, 1, 2 (2007).

I am amazed that these two papers were written a mere decade ago! My amazement about the anatomy paper is that the details have changed a lot just a few years later. My amazement about the data model paper is that nobody ever seems to learn anything from history. Lets talk about the data model paper first.

A decade ago, the buzz was all XML. Vendors were intent on adding XML to their relational engines. Industry analysts (and more than a few researchers) were touting XML as “the next big thing”. A decade later it is a niche product, and the field has moved on. In my opinion, (as predicted in the paper) it succumbed to a combination of:

- excessive complexity (which nobody could understand)
- complex extensions of relational engines, which did not seem to perform all that well and
- no compelling use case where it was wildly accepted

It is a bit ironic that a prediction was made in the paper that X would win the Turing Award by successfully simplifying XML. That prediction turned out to be totally wrong! The net-net was that relational won and XML lost.

Of course, that has not stopped “newbies” from reinventing the wheel. Now it is JSON, which can be viewed in one of three ways:

- A general purpose hierarchical data format. Anybody who thinks this is a good idea should read the section of the data model paper on IMS.
- A representation for sparse data. Consider attributes about an employee, and suppose we wish to record hobbies data. For each hobby, the data we record will be different and hobbies are fundamentally sparse. This is straightforward to model in a relational DBMS but it leads to very wide, very sparse tables. This is disastrous for disk-based row stores but works fine in column stores. In the former case, JSON is a reasonable encoding format for the “hobbies” column, and

several RDBMSs have recently added support for a JSON data type.

- As a mechanism for “schema on read”. In effect, the schema is very wide and very sparse, and essentially all users will want some projection of this schema. When reading from a wide, sparse schema, a user can say what he wants to see at run time. Conceptually, this is nothing but a projection operation. Hence, ‘schema on read’ is just a relational operation on JSON-encoded data.

In summary, JSON is a reasonable choice for sparse data. In this context, I expect it to have a fair amount of “legs”. On the other hand, it is a disaster in the making as a general hierarchical data format. I fully expect RDBMSs to subsume JSON as merely a data type (among many) in their systems. In other words, it is a reasonable way to encode spare relational data.

No doubt the next version of the Red Book will trash some new hierarchical format invented by people who stand on the toes of their predecessors, not on their shoulders.

The other data model generating a lot of buzz in the last decade is Map-Reduce, which was purpose-built by Google to support their web crawl data base. A few years later, Google stopped using Map-Reduce for that application, moving instead to Big Table. Now, the rest of the world is seeing what Google figured out earlier; Map-Reduce is not an architecture with any broad scale applicability. Instead the Map-Reduce market has morphed into an HDFS market, and seems poised to become a relational SQL market. For example, Cloudera has recently introduced Impala, which is a SQL engine, built on top of HDFS, not using Map-Reduce.

More recently, there has been another thrust in HDFS land which merit discussion, namely “data lakes”. A reasonable use of an HDFS cluster (which by now most enterprises have invested in and want to find something useful for them to do) is as a queue of data files which have been ingested. Over time, the enterprise will figure out which ones are worth spending the effort to clean up (data curation; cov-

ered in Chapter 12 of this book). Hence, the data lake is just a “junk drawer” for files in the meantime. Also, we will have more to say about HDFS, Spark and Hadoop in Chapter 5.

In summary, in the last decade nobody seems to have heeded the lessons in “comes around”. New data models have been invented, only to morph into SQL on tables. Hierarchical structures have been reinvented with failure as the predicted result. I would not be surprised to see the next decade to be more of the same. People seemed doomed to reinvent the wheel!

With regard to the Anatomy paper; a mere decade later, we can note substantial changes in how DBMSs are constructed. Hence, the details have changed a lot, but the overall architecture described in the paper is still pretty much true. The paper describes how most of the legacy DBMSs (e.g. Oracle, DB2) work, and a decade ago, this was the prevalent implementation. Now, these systems are historical artifacts; not very good at anything. For example, in the data warehouse market column stores have replaced the row stores described in this paper, because they are 1–2 orders of magnitude faster. In the OLTP world, main-memory SQL engines with very lightweight transaction management are fast becoming the norm. These new developments are chronicled in Chapter 4 of this book. It is now hard to find an application area where legacy row stores are competitive. As such, they deserve to be sent to the “home for retired software”.

It is hard to imagine that “one size fits all” will ever be the dominant architecture again. Hence, the “elephants” have a bad “innovators dilemma” problem. In the classic book

by Clayton Christiansen, he argues that it is difficult for the vendors of legacy technology to morph to new constructs without losing their customer base. However, it is already obvious how the elephants are going to try. For example, SQLServer 14 is at least two engines (Hekaton a main memory OLTP system and conventional SQLServer — a legacy row store) united underneath a common parser. Hence, the Microsoft strategy is clearly to add new engines under their legacy parser, and then support moving data from a tired engine to more modern ones, without disturbing applications. It remains to be seen how successful this will be.

However, the basic architecture of these new systems continues to follow the parsing/optimizer/executor structure described in the paper. Also, the threading model and process structure is as relevant today as a decade ago. As such, the reader should note that the details of concurrency control, crash recovery, optimization, data structures and indexing are in a state of rapid change, but the basic architecture of DBMSs remains intact.

In addition, it will take a long time for these legacy systems to die. In fact, there is still an enormous amount of IMS data in production use. As such, any student of the field is well advised to understand the architecture of the (dominant for a while) systems.

Furthermore, it is possible that aspects of this paper may become more relevant in the future as computing architectures evolve. For example, the impending arrival of NVRAM may provide an opportunity for new architectural concepts, or a reemergence of old ones.