The relational model is dead, SQL is dead, and I don't feel so good myself

Paolo Atzeni Christian S. Jensen Giorgio Orsi Sudha Ram
Letizia Tanca Riccardo Torlone

ABSTRACT

We report the opinions expressed by well-known database researchers on the future of the relational model and SQL during a panel at the International Workshop on Non-Conventional Data Access (NoCoDa 2012), held in Florence, Italy in October 2012 in conjunction with the 31st International Conference on Conceptual Modeling. The panelists include: Paolo Atzeni (Università Roma Tre, Italy), Umeshwar Dayal (HP Labs, USA), Christian S. Jensen (Aarhus University, Denmark), and Sudha Ram (University of Arizona, USA). Quotations from movies are used as a playful though effective way to convey the dramatic changes that database technology and research are currently undergoing.

1. INTRODUCTION

As more and more information becomes available to a growing multitude of people, the ways to manage and access data are rapidly evolving as they must take into consideration, on one front, the kind and volume of data available today and, on the other front, a new and larger population of prospective users. This need on two opposite fronts has originated a steadily growing set of proposals for non-conventional ways to manage and access data, which fundamentally rethink the concepts, techniques, and tools conceived and developed in the database field during the last forty years. Recently, these proposals have produced a new generation of data management systems, mostly non-relational, proposed as effective solutions to the needs of an increasing number of large-scale applications for which traditional database technology is unsatisfactory.

Today, it is common to include all the non-relational technologies for data management under the umbrella term of "NoSQL" databases. Still, it is appropriate to point out that SQL and relational DBMSs are not synonymous. The former is a language, while the latter is a mechanism for managing data using the relational model. The debate on SQL vs. NoSQL is as much a debate on SQL, the language, as on the relational model and its various implementations.

Relational database management systems have been around for more than thirty years. During this time, several revolutions (such as the Object Oriented database movement) have erupted, many of which threatened to doom SQL and relational databases. These revolutions eventually fizzled out, and none made even a small dent in the dominance of relational databases. The latest revolution appears to be from NoSQL databases that are touted to be non-relational, horizontally scalable, distributed and, for the most part, open source.

The big interest of academia and industry in the NoSQL movement gives birth, once more, to a number of challenging questions on the future of SQL and of the relational approach to the management of data. We discussed some of them during a lively panel at the NoCoDa Workshop, an event held in Florence, Italy in October 2012 organized by Giorgio Orsi (Oxford University), Letizia Tanca (Politecnico di Milano) and Riccardo Torlone (Università Roma Tre). We have used a provocative title (paraphrasing a quote often attributed to Woody Allen) and quotations from movies to elaborate on three main issues:

- the possible decline of the relational model and of SQL as a consequence of the rise of the non-relational technology,
- the need for logical data models and theoretical studies in the NoSQL world, and
- the possible consequences of sacrificing the ACID properties in favor of system performance and data availability.

In the following sections we discuss these issues in turn and close the paper with a final discussion. Since a consensus was reached on most of the issues addressed in the panel, we synthesize shared

opinions, rather than report contributions to the discussion by single individuals.

2. THE END OF AN ERA?

2.1 Relational databases

> "The ship will sink." "You're certain?"
> "Yes. In an hour or so, all of this will be at the bottom of the Atlantic."
> (Titanic. 1997)

According to Stonebraker et al., RDBMS are 25-year-old legacy code lines that should be retired in favor of a collection of from-scratch specialized engines [9]. Are we really attending the sinking of the relational ship?

One needs to distinguish between the relational model and its dominant query language, SQL, on the one hand and relational database management systems on the other.

The relational model and SQL were invented at a time when data management targeted primarily administrative applications. The goal was to support applications exemplified well by banking. The data is well structured: accounts, customers, loans, etc. And typical transactions include withdrawals and deposits that alter account balances. The relational model and SQL are well suited for managing this kind of data and supporting workloads made up from these kinds of transactions.

However, the data management landscape has evolved, and today's landscape of data management applications is much more diverse than it was when the relational model and SQL were born. Examples of this diversity abound: semi-structured data, unstructured data, continuous data, sensor data, streaming data, uncertain data, graph data, and complexly structured data. Similar diversities can be found in the workloads to be supported today.

Thus, while relational database systems were first proposed as a way to store and manage structured data, burgeoning NoSQL databases, such as CouchDB, MongoDB, Cassandra, and Hbase, have emerged as a way to store unstructured data and other complex objects such as documents, data streams, and graphs. With the rise of the real-time web, NoSQL databases were designed to deal with very large volumes of data.

Moreover, while relational database systems are usually scaled up (i.e., moved to larger and more powerful servers), NoSQL database systems are designed to scale out, i.e, the database is distributed across multiple hosts as load increases. This is more in line with real time web traffic as transaction rates and availability requirements increase and as data stores move into the cloud. The new breed of NoSQL systems are designed so they can easily scale up using low cost commodity processors to yield economic advantages.

Next, the data management applications have not just grown to concern more diverse kinds and uses of data. They have also become more complex. A single application may involve diverse kinds of data. This means that it is generally not possible for an application to use the single model and query language that is best for a single kind of data.

There are indeed two different issues here, related to the model level and to the implementation. In terms of implementation, it is clear (and it has been clear for more than a decade) that different applications have different requirements, especially when performance is a concern. This has led for example to separating OLTP and OLAP applications, even when the latter makes use of data produced by the former. Further, different engines with different capabilities have been developed for the two worlds, with specific support, the ones with more support for throughput of transactions and the others with support for very complex queries. With respect to models, the point is that most applications do need mainly simple operations over models that are somehow more complex than the relational one. NoSQL systems try to respond to these needs: implementations are new and specialized, operations are very simple, and diverse models (see the discussion on heterogeneity below) share the idea of being flexible (semistructured and with little or no schema).

2.2 SQL

> "Whoa, lady, I only speak two languages, English and bad English."
> (The Fifth Element. 1997)

A variety of data models and access methods are emerging and SQL is not suitable for any of them. Are we building the Babel Tower of query languages?

SQL has several advantages — it is a simple yet powerful declarative language for set-oriented operations. SQL captures the essential patterns of data manipulation, including intersections/joins, filters, and aggregations or reductions. Programmers who profess a dislike for SQL appear to have been deceived by its simplicity. The existence of languages such as SQLDF [4], which allows SQL queries on R data frames, add SQL functionality for analytics on Big Data. SQL's declarative expressions are

frequently more readable and compact than their R programmatic equivalents. Powerful extensions to SQL, based on window functions, provide a "split-apply" functionality otherwise known as map function. Combining these with SQL's GROUP BY operation, which is in reality a reduce function, essentially provides the equivalent of operations such as those in the Map Reduce framework.

However, in spite of the research and development, the relational model and SQL may not be the best foundation for managing every new kind of data and workload. The SQL-86 standard was a small and simple document. Then came SQL-89, SQL-92, SQL:1999, SQL:2003, SQL:2006, and SQL:2008. The current standard, SQL:2011, is very complex, and most data management professionals will find it challenging to understand. How many people have read and understood the entire SQL standard? Few claim that SQL is an elegant language characterized by orthogonality. Some call it an elephant on clay feet. With each addition, its body grows, and it becomes less stable. SQL standardization is largely the domain of database vendors, not academic researchers without commercial interests or users with user interests. Who is that good for?

Another aspect is that the SQL syntax requires the use of joins, considered ill-fit for, e.g., preferences and data structures for complex objects or completely unstructured data: many programmers would prefer to not do joins at all, keeping the data in a physical structure that fits the programming task as opposed to extracting it from a logical structure that is relational. Complex objects that contain items and lists do not always map directly to a single row in a single table, and writing SQL queries to grab the data spread out across many tables, when all you want is a record, is inconsistent with the belief that data should be persisted the way it is programmed.

On the other hand, the tumultuous developments we are observing have generated dozens of systems each with its own modeling features and its own APIs [2, 8], and this is definitely generating confusion. Indeed, the lack of a standard is a great concern for companies interested in adopting any of these systems [7]: applications and data are expensive to convert and competencies and expertise acquired on a specific system get wasted in case of migration. Efforts that support interoperability and translation are definitely needed [1]. Original approaches in this direction are needed, given the simplicity of operations and the almost total absence of schemas.

3. MODEL, THEORY AND DESIGN

3.1 Logical data models

> "Underneath, it's a hyper-alloy combat chassis, microprocessor-controlled. But outside, it's living human tissue: flesh, skin, hair, blood." (Terminator. 1984)

Aren't NoSQL database models too close to the physical data structures? What about physical data independence?

The ANSI SPARC architecture for database systems was defined in 1975 with the fundamental goal of setting a standard for data independence for DBMS vendor implementations. It appears that current NoSQL systems make no distinction between the logical and physical schema. Thus, the fundamental advantages of the ANSI SPARC architecture have been voided, which complicates the maintenance of these databases. Storing objects as they are programmed essentially negates the data independence requirement that then remains to be adequately addressed for NoSQL database systems. Strong typing of relations also allows definition of a variety of integrity constraints at the schema level, a very important consideration for transaction processing systems that support a variety of read, write, delete, and update transactions.

Relational database systems are criticized for the strong typing of relational schemas, which makes it difficult to alter the data model. Even minor changes to the data model of a relational database have to be carefully managed and may require downtime or reduced service levels. NoSQL databases have far more relaxed — or even nonexistent — data model restrictions. NoSQL Key Value stores and document databases allow applications to store virtually any structure it wants in a data element. Even the more rigidly defined BigTable-based NoSQL databases (Cassandra, HBase) typically allow new columns to be created with little effort. Actually, organizations should carefully evaluate the advantages and limitations of each type of systems (i.e. relational and NoSQL) for Big Data and then make an informed decision.

A common, high level interface could really be of use here. However it has to be simple, especially in terms of operations, as is the case for NoSQL systems. It is also worth mentioning that developers of the various systems follow "best practices" that support efficient execution of operations. An effort should be made to design a common interface by using the best practices of each system, with the goal of re-achieving physical independence.

3.2 Database theory

> "I've seen things you people wouldn't believe. [...] All those moments, will be lost in time, like tears in rain. Time to die." (Blade Runner. 1982)

Do we still need theoretical research in the new world? Has relational database theory become irrelevant?

The introduction of the relational model in 1970 marked a striking difference with respect to all the previous research on databases. The main reason for this lies in the strong mathematical foundations upon which this model is based, which provided the database research community with the possibility to approach the problems that were raised during the years by means of logical and mathematical tools, and to ensure the correctness and effectiveness of the proposed solutions by solid mathematical proofs.

This approach has caused the blooming of generations of splendid theoreticians who have set the foundations of the relational model, but have also contributed to adapting their experience to devise new methods and techniques for solving the problems derived from the advent of new challenges. Consider for instance the introduction of new paradigms for representing and querying semi-structured and unstructured data: since the nineties, invaluable theoretical research has laid the foundations for dealing with XML and the related query languages, with HTML Web data, with the Semantic Web, and with unstructured data like images and videos. It would be interesting to see what the work on semi-structured data and XML (modelling and languages) can contribute in the setting of NoSQL databases, since after all many of the problems rising from this new data model(s) have been discussed already within the semi-structured data research.

The lessons learned from developing the relational database theory have probably laid the methodological foundations for approaching most data-related problems, since, however unstructured and unkempt the datasets at hand, the understanding developed within the community will ever inform its research strategies.

3.3 Database design

> "They rent out rooms for old people, kill'em, bury'em in the yard, cash their social security checks." (No Country for Old Men. 2007)

How is database design affected by the recent paradigm shifts on logical data modeling? Is conceptual database design really too old for this country?

The methodological framework consisting of conceptual data modeling followed by the translation of the ER (or class-diagram) schema into a logical (relational) one can still be adopted: after all, these systems have to be accessed by applications. So, even if there is no schema in the data store, it is very likely that the data objects belong to classes, whose definitions appear in the programs, so some contribution could arise. At the same time, flexibility is a must, as objects could come from classes in an inheritance hierarchy, so polymorphism should be supported. The availability of a high-level representation of the data at hand, be it logical or conceptual, remains a fundamental tool for developers and users, since it makes understanding, managing, accessing, and integrating information sources much easier, independently of the technologies used.

4. ACID OR AVAILABLE?

> "Ask me a question I would normally lie to." (True Lies. 1994)

A relational database is a perfect world where data is always consistent (even if not true). Are the ACID properties really less relevant in modern database applications? Are we ready for a chaotic world where data is always available but only "eventually" consistent?

While preserving ACID properties may not be as important for databases that typically contain append only data, they are absolutely essential for most operational systems and online transaction processing systems, including retail, banking, and finance. ACID compliance may not be important to a search engine that may return different results to two users simultaneously, or to Amazon when returning sets of different reviews to two users. In these applications, speed and performance triumph the consistency of the results. However, in a banking application, two users of the same account need to see the same balance in their account. A utility company needs to display the same "payment due amount" to two or more users perusing an account. The idea of "eventual consistency" for such applications could lead to chaos in the business world. Is it by chance that just those applications that need full consistency are often those that better match the relational structure? Can we imagine a bank, a manufacturing or a commercial company which would rather use a complex-object data model to represent their data? This is probably why many

people mix up the structure of the relational model with the ACID properties, which in principle are completely independent aspects.

A consequence of the choices made in some systems about weak forms of consistency is that the burden is passed to applications developers, when they need to ensure more sophisticated transaction properties.

An observation that has been recently made about transaction management (and other implementation issues) is related to the fact that it can be easy to omit features, as this simplifies the development, but it might be difficult to reintroduce them later. Mohan [6] points out that there were experiences in the past with similar simplifications, and it was later very complex to obtain more general and powerful systems— some features needed to be rewritten from scratch.

5. FINAL COMMENTS

> "Look! It's moving. It's alive!!"
> (Frankenstein. 1931)

In spite of the shortcomings and inadequacies of the relational model and SQL, these technologies are, however, still going strong. Why? A key reason is that the systems that implement these are plentiful and have proven their worth. Perhaps the most important reason is that enormous investments are sitting in applications built on top of such systems. Companies around the globe rely on these applications and their underlying database management systems for their day-to-day business. Actually, relational DBMS provide the most understandable format for business application data, and at the same time guarantee the consistency properties that are needed in business. In addition, the skill sets of their current and prospective employees are targeted at these systems. It is not an easy decision to throw away relational and SQL technology and instead adopt new technology. Rather, it is much easier to extend the current applications and systems with no radical changes. Indeed, to the extent applications involve standard administrative data and "new" data, relational technology may even be best suited.

Thus, when is it reasonable for an organization to bet on a tool that is slightly incompatible with all the others, may be built by a community in open source model, does not grant consistency and concurrency control and is subject to change, neglect, and abandon at any point in time? The point is that there are killer applications – e.g. storing huge amounts of (read-only) social-network or sensor data in clusters of commodity hardware – that may make it worthwhile.

Therefore, we all believe that relational and NoSQL database systems will continue to coexist. In the era of large, decentralized, distributed environments where the amount of devices and data and their heterogeneity is getting out of control, billions of sensors and devices collect, communicate and create data, while the Web and the social networks are widening the number of data formats and providers. NoSQL databases are most often appropriate for such applications, which either do not require ACID properties or need to deal with objects which are clumsily represented in relational terms.

As a conclusion, NoSQL data storage appears to be additional equipment that business enterprises may choose to complete their assortment of storage services.

With all these questions ahead the contribution the database community can give is huge. Let us take a full breath and start anew!

6. REFERENCES

[1] P. Atzeni, F. Bugiotti, and L. Rossi. Uniform access to non-relational database systems: The SOS platform. In *CAiSE 2012*, Springer, pages 160–174, 2012.

[2] R. Cattell. Scalable SQL and NoSQL data stores. *SIGMOD Record*, 39(4):12–27, 2010.

[3] M. Driscoll. SQL is Dead. Long Live SQL! `http://www.dataspora.com/2009/11/sql-is-dead-long-live-sql/`, 2009.

[4] G. Grothendieck. SQLDF: SQL select on R data frames. `http://code.google.com/p/sqldf/`, 2012.

[5] G. Harrison. 10 things you should know about NoSQL databases. `http://www.techrepublic.com/blog/10things/10-things-you-should-know-about-nosql-databases/1772`, 2010.

[6] C. Mohan. History repeats itself: sensible and NonsenSQL aspects of the NoSQL hoopla. In *EDBT 2013*, ACM, pag. 11–16, 2013.

[7] M. Stonebraker. Stonebraker on NoSQL and enterprises. *Commun. ACM*, 54:10–11, 2011.

[8] M. Stonebraker and R. Cattell. 10 rules for scalable performance in 'simple operation' datastores. *Commun. ACM*, 54(6):72–80, 2011.

[9] M. Stonebraker, S. Madden, D. J. Abadi, S. Harizopoulos, N. Hachem, and P. Helland. The end of an architectural era: (it's time for a complete rewrite). In *VLDB 2007*, VLDB Endowment, pag. 1150-1160, 2007.