# TEMPORAL DATA MANAGEMENT IN NOSQL DATABASES

*Morgan D. Monger*[1]
*Ramon A. Mata-Toledo*[2]
*Pranshu Gupta*[3]

## ABSTRACT

*Temporal data management is recognized concept of managing data that changes over time. Traditionally, a temporal data management system (TDMS) was considered separate system, i.e. an extension of the traditional relational database management system (RDBMS). Recently a non-relational database, called NoSQL, has become quite popular for scalability and storage of big data. Even with this new approach for managing data, temporal data management is still a work in progress. This paper provides an overview of temporal data management and NoSQL databases. Also, temporal data management is discussed within the context of NoSQL databases. This paper concludes with observations and suggestions for future research.*

**Keywords: temporal database, NoSQL, database, big data, bi-temporal data**

## 1. INTRODUCTION

In general, the traditional relational databases management systems cannot support either historical queries or trend analysis. In today's business environment, both historical queries and trend analysis have become a necessity due to the fact that organizations need to adapt quickly and respond to any change in the economy or the ever-changing needs or taste of the customers. For any business or organization to adapt or anticipate to the aforementioned changes it is necessary to have good querying and reporting capabilities [Snodgrass, 1987]. A TDMS can provide easy access to historical queries and trend analysis because it manages time-variant data [Mata-Toledo and Monger, 2008]. Thus, there is need for temporal support in the corporate world. The decrease in storage costs and new mass storage technologies have augmented interest in TDMSs. Initial attempts to manage temporal data consisted of extending the relational database model. For example, TSQL2, temporal structured query language was defined as an extension of SQL2 where the underlying model was the bi-temporal conceptual data model [Snodgrass, 1986]. By bi-temporal data we mean data that can be tagged with transaction and valid timestamps to represent when the data was valid within the database and when it was valid in reality. Another attempt to create a TDMS was the static "Ingres" DBMS which extended the query language TQuel which, in turn, was a superset of Quel [Snodgrass, 1987]. A particular commonality of all these initial approaches was that all applications were ad hoc. That is, it was left up to the application designers and developers to discover, design, program and implement the temporal concepts that were required by the particular industry. Lately, time variants have been added to many data models but primarily to the object-oriented and relational models

1 Lead Developer/Designer, Datatel Inc., Fairfax, VA 22033
2 Department of Computer Science, James Madison University, Harrisonburg, VA 22801
3 PhD Candidate, Computing and Information Sciences, Kansas State University, Manhattan, KS 66502

(Bertino, Ferrari, and Guerrini, 1996 and Navathe and Ahmed, 1989). Even though temporal data management is critical to industries such as medical, insurance, scientific databases or airlines reservation systems and more, there is still lack of agreement on a standardized TDMS for good management of temporal data.

Some of the critical issues that need to be addressed by the underlying DBMS to better support the management of temporal data are scalability and the handling of "Big Data". The latter requires that the DBMS be able to control volume, variety and velocity of data while maintaining ACID (atomicity, consistency, isolation and durability) compliance [Kelly, 2012]. In addition to this, it is necessary to have a flexible data model. By this term we mean a model that provides the ability to define and use any number of custom attributes and relationships.

At the turn of the last century, a new class of databases, known as NoSQL- Not Only SQL - DBMSs have emerged. This type of DBMSs purposely breaks from the relational model with increasing focus on scalability and big storage [Pokorny, 2011]. NoSQL DBMSs are gaining popularity due to their simpler data access, easy deployment, and performance gains. In particular, NoSQL DBMSs can provide cell-level security in Big Data -a crucial aspect in the compliance of security and privacy regulations - of a concurrent population with a large number of users [Kelly, 2012]. In the next sections, we describe briefly the current state of the art of temporal data management and NoSQL databases before bridging these two concepts.

## 2. TEMPORAL DATA MANAGEMENT

Managing temporal data management is very difficult when conventional data models and query languages are used because the database application developers are, in general, responsible for complying with the time-varying nature of the data in an ad hoc manner that cannot be easily generalized to other critical applications within the database [Jensen and Snodgrass, 1999]. A TDMS manages temporal data similar to how a relational database management system (RDBMS) manages normal data. Temporal relationship operators and reference intervals are additional features generally used within a TDMS [Allen, 1983]. Table 1 depicts some of the various temporal relationship operators and examples of each. These temporal operators are quite powerful for querying data over specific time periods. To manage the different aspects related to time-variant data several approaches have been proposed over the years. These approaches include the use of middleware, query language extensions and the use of a native TDMS [Vassilakis, Georgiadis and Sotiropoulou, 1996]. In recent years, Oracle and MySQL have been evaluated for temporal data management implementations [Mata-Toledo and Monger, 2008 and Vicknair, Wilkins, and Chen, 2012]. For example, Figure 1 shows the structure for an Oracle-based TDMS. The implementation contains database triggers and procedures that allow bolt-on functionality without affecting the underlying RDBMS. Most modern approaches follow a similar pattern where the TDMS is not fully integrated into the RDBMS.

| Relation | Example |
|---|---|
| X *before* Y | XXX YYY |
| X *equal* Y | XXX<br>YYY |

| | |
|---|---|
| X *meets* Y | XXXYYY |
| X *overlaps* Y | XXX<br>YYY |
| X *during* Y | XXX<br>YYYYYYY |
| X *starts* Y | XXX<br>YYYYY |
| X *finishes* Y | XXX<br>YYYYY |

Table 1: Allen's Temporal Relationship Operators [Allen, 1983]

## 3. NOSQL DATABASES

NoSQLs are highly scalable database designed to address the issues of low potential of data access. As mentioned earlier, NoSQL databases abandon the relational model for performance gains. From a traditional point of view, NoSQL is just a giant repository of data. While relational DBMS are purposely designed to maintain tight

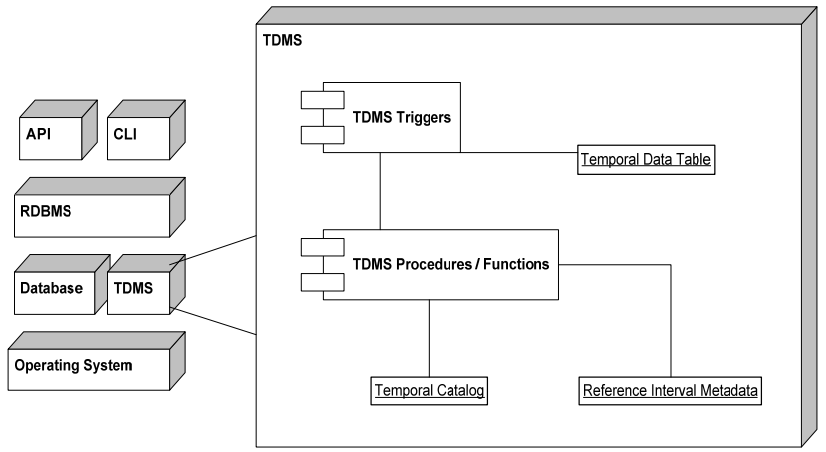


Figure 1: TDMS Structure [Mata-Toledo and Monger, 2008]

control over the data, NoSQL provides loose controls over data sets. In this respect relational systems use metadata to control data types and synchronized processes control to maintain data integrity. However, NoSQL uses loosely synchronized process to scatter data across many systems, with little regard for the quality of data stored at the servers. The advantages of NoSQL are reading and writing quickly, scalability, and mass storage support at a low cost. On the other hand, one of the perceived disadvantages of NoSQL, as the name indicates, is that it does not support SQL [Jing, Haihong, Guan and Du, 2011]. However, on this issue SQL versus NoSQL, there is a great divide among the practitioners; some of which do not favor the generality of SQL – as it applies to a large volume of data. For example, some argue that joins on large volume of data are "slowing the system down to a crawl". These practitioners see that as one of the main drawbacks of SQL citing as prime examples Google™ and Amazon™ which unsatisfied with the lack of response found in SQL decided to develop their own non-relational systems [Cogswell, 2012]. However, others practitioners favor that NoSQL can be designed to meet a wide variety of challenges

such as handling of large data set almost at real time pace [Cogswell, 2012 and Lane, 2012]. Other arguments favoring or opposing SQL versus NoSQL refer to the number of records retrieved as a result of a query. Non-relational databases provide "record-at-a-time" access instead of a normal SQL interface retrieving multiple records [Stonebraker, 2010]. Some implementations still support bulk reads, but limit the number of documents returned for performance reasons. Another performance consideration is physical scaling. Traditional relational databases rely on vertical scaling, which means scaling on a single machine. NoSQL databases offer shared-nothing horizontal scalability where additional nodes can be added for performance gains [Cattell, 2011]. Figure 2 demonstrates the difference between horizontal and vertical scaling. NoSQL databases still contain the overhead of disk storage and multithreaded access, which are found in the RDBMS counterparts [Stonebraker, 2010]. The largest performance boost comes from abandoning ACID transactions [Cattell, 2011]. NoSQL databases instead implement a "BASE" pattern, which stands for Basically Available, Soft state, Eventually consistent. Eventual consistency is an interesting shift where updates are eventually propagated to all nodes [Cattell, 2011].

NoSQL databases tend to fall into one of three categories: key-value stores, document stores, or column-oriented stores [Schindler, 2012]. Key-value stores are the most basic type of storage where a fixed key is associated to a value. Document stores are similar to key-value stores, but they also understand the format of the stored value, such as XML [Schindler, 2012]; the later examples focus on this type of NoSQL database. Column-oriented stores are typically for large-scale data and can associate multiple columns to a fixed key [Schindler, 2012]. Popular implementations include memcached, Google™ BigTable, MongoDB™, RavenDB™, and Amazon™ Dynamo [Cattell, 2011]. Other practitioners include under the NoSQL umbrella other non-relational databases such as XML databases, object-oriented databases, and graph databases [Pokorny, 2011].

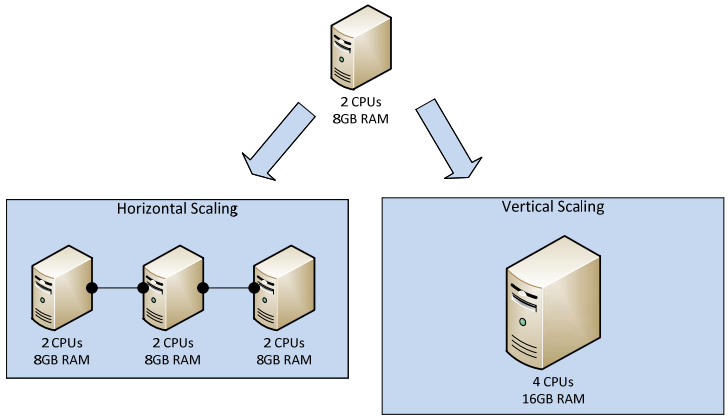


Figure 2: Horizontal vs. Vertical Scaling

## 4. TEMPORAL DATA MANAGEMENT IN NOSQL DATABASES

In this section we will the consider some of the basic aspects of managing temporal data within NoSQL databases. For the purposes of this paper we will utilize, as an example,

RavenDB™ which is a NoSQL document store with strong formatting using JavaScript Object Notation (JSON). Under normal circumstances, a query for a particular record such as “John Smith” would result in a single document returned from the NoSQL document store. With temporal data in mind, the NoSQL database should return a set of “John Smith” documents that show the changes over time. For simplicity, the example will focus on document-level timestamps rather than property-level.

Figure 3 depicts a collection of JSON objects returned for a single user with bi-temporal properties. In this example, the 2011 and 2012 salaries are tracked for John Smith. The transaction and valid time differ because they track different aspects, period of database validity versus real world validity.

Aside from the bi-temporal properties, the NoSQL database can be modified to accept temporal selection criteria. The selection criteria can be based on Allen’s temporal operators (shown in Table 1) and should work even in the absence of the relational model. Figure 4 demonstrates how a temporal query could appear against a modified RavenDB™. In this example, the specified dates cover the range from October 1st to October 31st and the operator specifies that the resulting documents must match within this date range.

## 5. SECURITY ISSUES IN NOSQL DATABASES

That secure systems are necessary in today business environment is an understatement due to the continuous external threats and malicious attacks that occur daily on computer systems. In the area of databases in general is not any different. However, when it comes to NoSQL, as mentioned before, each flavor of NoSQL aims to meet a particular purpose such as handling very large datasets, redundancy, metrics gathering, or focusing on specific type of content. With this in mind, it is easy to see that security was never the primarily goal and, as stated in [Lane, 2012], “never was, it’s not day, and won’t be for some time.” Nevertheless, we should not infer from this that security is totally absent from NoSQL. On the contrary, all NoSQL providers such as Couch™ and MongoDB™ do have some security designed to address specific threats [Lane, 2012]. The current practice is to


```
[
  {
    "Name": "John Smith",
    "Salary": "85,000",
    "StartTransactionTime": "2011-11-12T13:19:28.0000000Z",
    "EndTransactionTime": "2012-10-30T13:07:39.0000000Z",
    "StartValidTime": "2011-01-01T00:00:00.0000000Z",
    "EndValidTime": "2011-12-31T11:59:59.0000000Z"
  },
  {
    "Name": "John Smith",
    "Salary": "100,000",
    "StartTransactionTime": "2012-10-30T13:07:39.0000000Z",
    "EndTransactionTime": null,
    "StartValidTime": "2012-01-01T00:00:00.0000000Z",
    "EndValidTime": "2012-12-31T11:59:59.0000000Z"
  },
]
```

Figure 3: Bi-Temporal Data in JSON

```
GET /indexes/dynamic/Person?query=Salary:[During("2012-10-
01T00:00:00.0000000Z",("2012-10-311T00:00:00.0000000Z")] AND Name:"John
Smith"&start=0&pageSize=128 HTTP/1.1
Accept-Encoding: deflate,gzip
Content-Type: application/json; charset=utf-8
Host: 127.0.0.1:8081
```

Figure 4: Temporal NoSQL Query

host NoSQL databases in protected environments that include firewalls and strict access controls. However, as stated in [Lane, 2012] "the irony is that just about every application platform designed specifically for use with NoSQL systems is Web-centric." As stated in [Shimel, 2012], that NoSQL does not have enough security mechanisms as the security experts would like which is primarily due to the fact that "customers are not necessarily asking for security per se." In this respect, priority to satisfy customer needs has taken precedence and, as indicated also by [Shimel, 2012], "If the customers are paying, you are building. If it is not important to them, it not important to you." With this attitude, it is obvious that additional infrastructure needs to be developed to provide the required database security.

A question that one could ask is the following "Are customers neglecting security in general or is there a particular type of security that they are interested? The answer to this question was stated early in the paper when we mentioned "cell-level security in Big Data". There are systems such as Accumulo™ that make emphasis on the ability to assign user access permissions down to the cell-level. That is, users are given access to the database and all tables within it but with some cells encrypted with highly proved encrypted mechanisms. Notice that by encrypting those cells, the database administrators are forced to comply with privacy and security at the company, state, and federal regulations.

## 6. CONCLUSIONS

Temporal data is data that changes over time and can be managed by a TDMS. Temporal data exists in both SQL and NoSQL databases, so there should be a TDMS approach that works for each. Though NoSQL databases avoid the relational model for performance gains, NoSQL databases benefit from the better temporal data support, Big Data management and scalability. With the increase in time-variant data in both types of databases, there is need for a generalized approach for a TDMS for both SQL and NoSQL databases.

The proposed modifications mentioned earlier can provide a reasonable foundation for querying temporal data in NoSQL databases. Temporal XQuery queries against NoSQL XML document stores and true XML databases could be a good research area which the authors intend to explore.

## REFERENCES

Allen, J.F., "Maintaining knowledge about temporal intervals," *Communications ACM*, vol. 26, 1983, 832-843.

Cattell, R., "Scalable SQL and NoSQL data stores" *SIGMOD Rec.* 39, 4, May 2011, 12-27.

Gao, D., and Snodgrass, R.T., "Syntax, Semantics, and Query Evaluation of the tXQuery Temporal XML Query Language," *TimeCenter Publications., Aalborg, Denmark*, Tech. Rep. TR-72, March 2003.

Mata-Toledo, R. and Monger, M., "Implementing a temporal data management system within oracle." *Journal of Computing Sciences in Colleges* 23, 3, January 2008, 76-81.

Pokorny, J., "NoSQL databases: a step to database scalability in web environment." In *Proceedings of the 13th International Conference on Information Integration and Web-based Applications and Services* (iiWAS '11). *ACM*, 2011, 278-283.

Schindler, J., "I/O characteristics of NoSQL databases." *Proc. VLDB Endow.* 5, 12, August 2012, 2020-2021.

Stonebraker, M., "SQL databases v. NoSQL databases." *Commun. ACM* 53, 4, April 2010, 10-11.

Vassilakis, C., Georgiadis, P. and Sotiropoulou, A., "A Comparative Study of Temporal DBMS Architectures," *in DEXA Workshop*, 1996, 153-164.

Vicknair, C., Wilkins, D., and Chen, Y., "MySQL and the trouble with temporal data.", In *Proceedings of the 50th Annual Southeast Regional Conference (ACM-SE '12)*, 2012, 176-181.

Snodgrass, R., "The temporal query language TQuel." *ACM Trans. Database Syst.* 12, 2, 1987, 247-298.

Jensen, C. S., & Snodgrass, R. T., "Temporal data management." *Knowledge and Data Engineering, IEEE Transactions on*, 11(1), 1999, 36-44.

Jing H., Haihong, E., Guan L., Jian D., "Survey on NoSQL database." *Pervasive Computing and Applications (ICPCA), 2011 6th International Conference on*, 2011, 363-366.

Kelly J., "Accumulo: Why The World Needs Another NoSQL Database." *Big Data*, Aug 2012.

Batini, C., Ceri, S., and Navathe, S. B., *Conceptual database design*. Benjamin/Cummings, 1992.

Snodgrass, R. T., "Temporal databases." In *IEEE computer*, 1986.

Bertino, E., Ferrari, E., & Guerrini, G., "A formal temporal object-oriented data model." *Advances in Database Technology—EDBT'96*, 1996, 342-356.

Navathe, S. B., & Ahmed, R., "A temporal relational model and a query language." *Information Sciences*, *49*(1), 1989, 147-175.

Cogswell, Jeff., "SQL vs. NoSQL: Which Is Better?" [Internet], July 2012, Available from http://slashdot.org/topic/bi/sql-vs-nosql-which-is-better/

**L**ane A., "A Response To NoSQL Security Concerns" [Internet], Feb 6, 2012, Available from http://www.darkreading.com/blog/232600288/a-response-to-nosql-security-concerns.html

Shimel, A., "Is Security An Afterthought For NoSQL?" [Internet], Jan 25, 2012, Available from http://www.networkworld.com/community/blog/security-afterthought-nosql