computing@computingonline.net
www.computingonline.net

Print ISSN 1727-6209
On-line ISSN 2312-5381
International Journal of Computing

IMPLEMENTING BI-TEMPORAL PROPERTIES INTO VARIOUS NOSQL DATABASE CATEGORIES

Mohammed Eshtay [1], Azzam Sleit [1], Monther Aldwairi [2,3]

[1] University of Jordan, Jordan, m.eshtay@fgs.ju.edu.jo, azzam.sleit@ju.edu.jo
[2] Jordan University of Science and Technology, Jordan, munzer@just.edu.jo
[3] Zayed University, United Arab Emirates, monther.aldwairi@zu.ac.ae

Paper history:
Received 27 September 2018
Received in revised form 4 January 2019
Accepted 7 March 2019
Available online 31 March 2019

Keywords:
NoSQL Databases;
Bitemporal Properties;
Cassandra;
Redis;
Column Oriented Stores;
Key-value Stores.

Abstract: NoSQL database systems have emerged and developed at an accelerating rate in the last years. Attractive properties such as scalability and performance, which are needed by many applications today, contributed to their increasing popularity. Time is very important aspect in many applications. Many NoSQL database systems do not offer built in management for temporal properties. In this paper, we discuss how we can embed temporal properties in NoSQL databases. We review and differentiate between the most popular NoSQL stores. Moreover, we propose various solutions to modify data models for embedding bitemporal properties in two of the most popular categories of NoSQL databases (Key-value stores and Column stores). In addition, we give examples of how to represent bitemporal properties using Redis Key-value store and Cassandra column oriented store. This work can be used as basis for designing and implementing temporal operators and temporal data management in NoSQL databases.

1. INTRODUCTION

Relational database management systems (RDBMS) were and still dominant in the market of database management systems because of the services they provide such as transaction processing and the well-established structure. RDBMS apply the same relational model. All of them use the SQL language and they differ by the enhancements they provide Cattell (2011). Despite the popularity of RDBMS, they faced a set of challenges due to the wide spread of the INTERNET, the emergence of many fields such as social networks and the vast amount of data to be handled. RDBMS performance and scalability are two important properties that were not adequate for many of the new distributed applications. In many cases, the complexity of ACID (Atomicity, Consistency, Isolation, Durability) design aspect of RDBMS to guarantee the transaction reliability is not required in some applications and can be passed for the sake of other aspects such as the performance [1]. For example, in the case of social networks, we need some sort of high scalability with high degree of structure flexibility and set of simple operations [2, 3]. Due to these needs, many systems emerged to support scalability. They depend on a set of simple operations and do not follow the strict relational databases design principles. These systems are called NoSQL (Not only SQL) [3, 4]. The term NoSQL is generally used to refer to non-relational databases. It describes the distributed no. relational databases that emerged to deal with the huge amount of data generated by the Web 2 applications [5]. In the age of data, big data played an essential role in pushing the need for NoSQL and in the growing popularity of such systems. The flexible data models offered by these systems in contrast to the strict rigid structure of RDBMS and the continues need for data availability encouraged the use of NoSQL databases [3]. Google was the leader in adopting these systems by instituting BigTable [6] in 2006, followed by Dynamo [7], which was introduced by Amazon in 2007. Properties such as the ability to scale rapidly, performance, continuous availability and partition tolerance overcome the historical satisfaction of relational database model. These

properties increase the interest in NoSQL database. Furthermore, they cause fast and huge development because of the attractiveness of the model for many of the new applications [8]. On the contrary, this fast development led to the emergence of the problem of heterogeneity. Many NoSQL database developed and each of which has its own development API to offer for the user. We also note that the majority of these systems do not support or discuss the bitemporal properties that are needed in many applications [9]. The need for temporal data management is seen important in many well-known applications such as insurance, airline ticket reservations, medical applications and more. The temporal data management is discussed from many angles in the RDBMS, and temporal databases are considered as an extension of RDBMS. On the other hand, the temporal properties still need much work to be implemented and used in NoSQL databases [1] specially with the emerging applications that produce huge amount of timestamped data, such as sensor data, financial tickers and e-commerce [10]. Moreover, the relation between temporal data and NoSQL databases is not yet well configured and the door is open for more research to make NoSQL systems benefit from the bitemporal properties. In order to solve the aforementioned issues, this paper discusses bitemporal properties and proposes different variants to help embed the bitemporal characteristics in NoSQL databases. Since NoSQL databases are heterogeneous and conform to different structures, these variants are dealing with the problem taking into account the unique properties of each kind of NoSQL databases categories. The models will cover Key-value stores and Column Oriented Stores. We begin with a discussion of the principal features of NoSQL databases (Section 2). In Section 3, we describe concepts that are essential to understanding temporal data management. Section 4 proposes how we can embed temporal data into NoSQL databases. Section 5 concludes the paper.

2. NOSQL SYSTEMS

The main reason for the popularity of the NoSQL systems is their appropriateness for the new type of applications that require strong support of scalability, good performance, and big data management. NoSQL databases overcome the limitations we faced in RDBMS such as the growth of the volume of big data, semi-structured data and the increased data connectivity. In RDBMS, the data must be structured and must follow the strict model of the relational model [9]. In contrast to RDBMS, NoSQL databases have the ability to scale horizontally besides the vast amount of data that is handled by these systems. NoSQL databases are divided into four main categories: Key-value stores, Document stores, Column Oriented Stores, and Graph Databases [11, 12].

2.1 KEY-VALUE STORES

The first and the simplest category of NoSQL databases is Key-value stores. In this type, the data is organized in a form of map; each value is attached to a unique key. A key identifies a value and is used for entering, retrieving and deletion of data. The values in these systems are schema flexible. These values that are attached to the keys might be of any kind, they might be simple such as strings and they might be of complex types such as arrays and lists [2] (see Fig. 1). Key-value databases can be imagined as a relation table with two fields: key and value. The value field can take data of any type and of any length, which make Key-value databases suitable for unstructured data (typically BLOB). These systems focus the attention on scalability rather than consistency [13]. In addition, the simplicity of their data model structure makes the data retrieval very quick, which is appropriate for big data processing supporting scalability and availability characteristics. Some the famous Key-value stores are: Amazon dynamo [7], LinkedIn Voldemort [14], Redis [15] and Basho Riak [16].

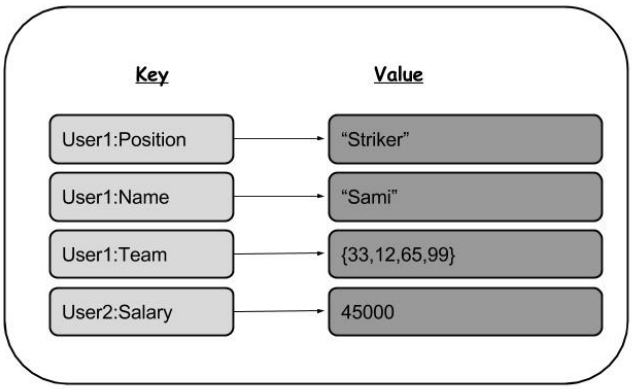

Figure 1 – Key-value NoSQL Database

2.2 COLUMN ORIENTED STORES

In traditional RDBMS, the data is organized in tables that consist of a collection of rows. Each row is identified by a unique id to be used for lookup process. Whereas column oriented NoSQL databases focus on columns to store data rather than rows. Column oriented databases tackle aspects such as the big number of columns and the schema changes. Each column has an indexed unique key to facilitate data retrieval and only columns specified by the query are needed to be retrieved, which will minimize the I/O cost. Unlike RDBMS that stores

rows contiguously, these systems store column values contiguously. Therefore, adding new Column oriented databases store data in rows each of which is identified by a primary key. The components of the row are a set of column families that contain the values of the given row. Furthermore, we can add one more level of grouping using super columns. Super column acts as a key for one or more columns that it holds, Fig. 2. Comparing to other NoSQL categories column stores databases provide better indexing and querying capabilities than Key-value stores. On the other hand, column stores face the same limitation of Key-value stores that any logic requiring relations must be implemented in the client application [17]. HBase [18], Cassandra [19] and Accumulo [20] are examples of column stores database.

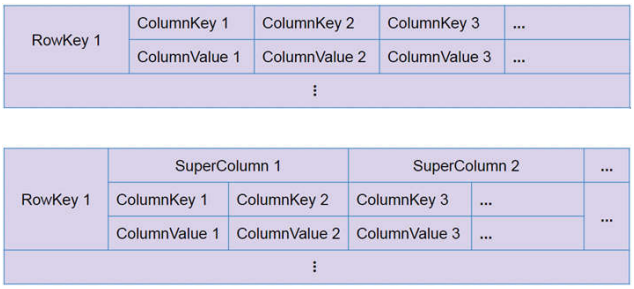

Figure 2 – Column Stores

2.3 DOCUMENT ORIENTED STORES

Another type of NoSQL is document based databases. As the name indicates, these databases are designed to manage documents. Documents are used to store data using some standard data exchange format such as JSON (JavaScript Option Notations) or BSON (Binary JSON). The main idea of document based databases is to provide big data storage and good query performance. Documents contain semistructured data in the form of attribute name/value pairs. The data is represented as documents and organized in a hierarchical way. Each document contains set of nested fields and list of attributes attached to the identifier that is used for indexing and retrieving [12] as seen in Figure 3. Documents dont adhere to a fixed schema; in the contrary, they allow flexible schema in which documents may contain subdocuments and documents as lists [4, 10]. Document stores are useful when data is suitable for representation as documents i.e. blogging sites or content management systems (CMS). The main problem of the document stores is the absence of built-in relationships between documents. MongoDB [21] and Apache CouchDB [22] are examples of this type.

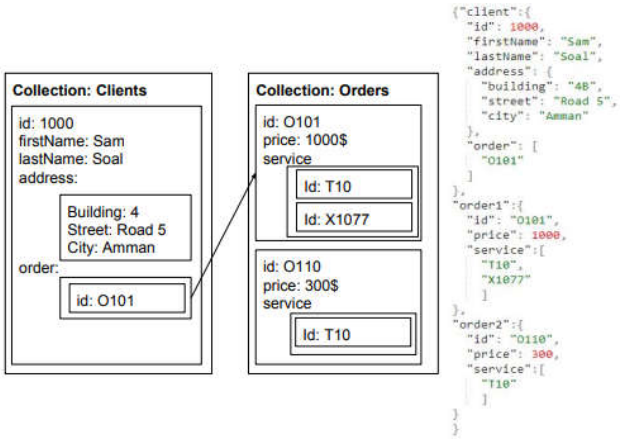

Figure 3 – Document Stores

3. TEMPORAL DATA MANAGEMENT

Generally, traditional databases do not deal directly with historical data and temporal characteristics. In many cases, the handling of the temporal properties is left to the developers. They find, analyze and handle such data taking into account the specific properties of the application [1]. The relational database systems that provide built-in time management for data are called temporal databases. Temporal databases support two types of time properties, valid time and transaction time. Valid time represents the validity interval of the data, which is the period that the data is considered true. The interval consists of start and end times. On the other hand, transaction time is the time when data stored in the database. The combination of both valid time and transaction time forms bitemporal properties [23]. Many researches have been conducted on the relation between the RDBMS and bitemporal data. For example, TSQL2 is an extension of SQL that handles the bitemporal properties and supports many temporal operators. TDBMS tends to handle the temporal data in the same way relational databases handle the normal data. Nowadays, big data is one of the topics that receive great interest in databases research [1]. Handling temporal data in big data systems raises huge research challenges and many opportunities for enhancements [24]. The main reason behind this potential is that the previous techniques, tools and algorithms developed to deal with bitemporal data in preceding technologies such as RDBMS and TDBMS in the foregoing years are inappropriate for big data. In the case of big data, we are concerned with the main characteristics of big data such as scalability, performance, heterogeneity, and volume. The line of the research is concerned with exploiting

the temporal properties of the big data model [24]. The implementation of bitemporal data management usually involves implementing temporal operators such as before, overlap and coalesce operations as additional features [25]. Some work has been done to manage the temporal characteristics in different NoSQL databases. Hu and Dessloch in [26] presented a definition for temporal operators such as union, intersection, and filter. They applied it to column-based NoSQL databases (CoNoSQLDBs). Their work has been presented to overcome the problem of implicit history representation (IHR) which is originally implemented in CoNoSQLDBs. Another NoSQL document-based database implemented temporal characteristics is MarkLogic [27]. They introduced set of functions to insert, update and delete temporal documents. In addition, they defined a set of terms such as instant, period, valid time and system time. NoSQL databases systems are platforms developed with big data needs in mind. NoSQL databases are the best place to embed and exploit bitemporal properties. The discussion of the relation between NoSQL databases and temporal characteristics still needs more effort and research. In this paper, we will discuss how to embed bitemporal properties in different NoSQL systems.

4. EMBEDDING TEMPORAL CHARACTERISTICS

The current NoSQL databases do not offer built-in temporal data management, which is needed by many applications and useful in many situations [26]. The idea of this section is to present different ways to extend different categories of NoSQL databases in a way that allow them to handle bitemporal data properties. The existed techniques, tools, and algorithms that manage the temporal data in the relational databases handle these data the same way the handle normal data, which cannot be applied directly to the new generation of databases. Non-relational databases have different lenient data model to deal with semi-structured data compared to the relational model which has a strict rigid model. In this section, we will introduce with case studies how we can deal with temporal information in various NoSQL databases.

4.1 KEY-VALUE

In Key-value stores, the data is stored as a Key - Value pairs. The values are referenced by the keys. In the case of bitemporal data, we need to manage the valid and transaction times. We will propose set of ideas to help in the management of temporal data in Key-value stores. One of the solutions is to add two attributes in each Key-value pair, one for the valid time in the form of validTimeStart and validTimeEnd attributes as shown in the below example, Table 1.

Table 1. Embedding temporal data into Key-value Stores

ROW		
ID=1	**FirstName**	Moe
	LastName	Sami
	Salary	1000
	ValidTimeStart	24-1-2016
	ValidTimeEnd	24-4-2017
	TransactionTime	1-1-2016

In the example, we can see that the key is 1 and it refers to the set of values (FirstName, LastName, Salary, and ValidTime). The salary is a field of data that is affected by the time. In order to keep temporal property of the field salary, we need to manage the valid time, in each inserted Key-value pair we need to keep track of transaction time and the valid time. Transaction time could be part of the key so we will be able to retrieve chain of time series information. A clearer and more structured method leads us to another solution. The idea is to divide the key into two parts to form a composite key. This composite key consists of a part that identifies the row, which is the regular key and the other one represents the interval of the valid time. In such a way, we can make the key suitable for retrieving data according to the validity of the information as depicted in Fig. 4.

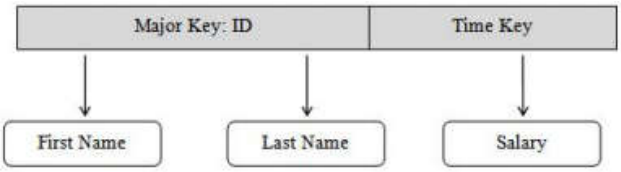

Figure 4 – Embedding Temporal Data in Key-value Stores

Redis [15] is a NoSQL database that is not only supports simple Key-value structure in which a string key associated with a string value but can also be associated with a complex structure. Redis supports various types of data structures, which gives us wide flexibility to use them to manage temporal data. The simplest are the string values associated with a key. Lists, sets, sorted sets and hashes are also value data structures available too. Redis does not support built in temporal

management but it offers set of functions that can be used to achieve this purpose. In the below example we will use Radis hash to store the values of the first and the last name of the employee and the history of his payroll record. The structure of our Key-value data is:

<majorKey><timeKey><dataValue>.

```
HMSET 1 firstName "John" lastName "Smith"
HMSET 1:2014105 salary 1000 vs 2014101 ve 2015121
HMSET 1:20151210 salary 2000 vs 2015122 ve 201751
HMSET 1:2017051 salary 3000 vs 201752 ve +inf
```

The first statement creates an employee with a major key of value 1. The next three statements add the salary of the employee and the period of the validity of this salary. Each one is referred to using the key (major and time). The result of the executing this code is depicted in Figure 5. One of the ways to get the history of the employee with major key 1 is to do something like:

```
keys 1:*
```

which will return

A. "1:20151210"
B. "1:2014105"
C. "1:201751"

Redis also offers sorted set that can be used to store values. In such a case, we can use timestamps to mark the historical data that we want to deal with. In this case, we can implicitly specify the valid start and end times. The valid start time is the current timestamp and the valid end time is the timestamp of the next value in the set. In the below example, we show the structure of the data and we use Redis to store and fetch set of historical data:

```
zadd StockPrice 1 11.1
zadd StockPrice 2 10.9
zadd StockPrice 3 10.5
zadd StockPrice 4 10.8
zadd StockPrice 5 12.15
zadd StockPrice 6 11.99
zadd StockPrice 7 11.9
```

To retrieve subset of the values:

```
zrangebyscore historical 2 4
```

It returns:

10.9, 10.5, 10.8

The aforementioned ideas will handle the temporal properties using the available capabilities of the existing NoSQL. It would be more efficient to have specific time series type. This type would be similar to a hash with internal handling of bi-temporal properties and specific indexing capabilities.

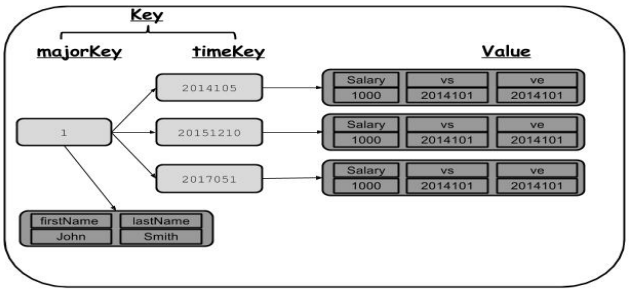

Figure 5 – Redis representation of temporal data

4.2 COLUMN ORIENTED

Column oriented NoSQL databases are inspired by big table. The main idea is to have structured data that can scale to large size. They have three main components the key, the column family and the super column. Row keys are the main object for data distribution and partitioning. This row key property is going to be exploited in the model that we propose for temporal data management. Values belonging to the same column are stored separately on the same disk. Columns are grouped in super columns [17, 28]. The management of temporal data is going to be different in this type since column oriented databases attach the timestamps to the saved values. In each column, multiple data values are stored and sorted by their time stamps. One way is to use these timestamps to handle valid time or transaction time; actually, timestamps are more appropriate to represent transaction time. Furthermore, maintaining valid time can be done by adding additional columns to express the valid time interval. Adding two columns is trivial and straightforward solution, the first column will represent the valid time start and the second one is to represent valid time end. Table 2 shows an example of such data. In this example, we can track the salary of the employee using the validation time stored in the added two columns.

Table 2. Column Oriented

	Name	Salary	vtFrom	vtTo	Tt
ID: 1	Sami	30000	1	2	1/1/2014
	Sami	40000	3	5	1/6/2015
	Sami	50000	5	∞	10/10/2016

In the following, we will use Cassandra [19] which is a well-known NoSQL column store to show the ability to model temporal data in column oriented databases. Cassandra is open source distributed NoSQL system to handle the huge amount of data across many servers. Cassandra has introduced its query language, which is called Cassandra Query Language (CQL) to work as an interface for accessing it.

```
CREATE TABLE Employee (
    id INT,
    name TEXT,
    salary FLOAT,
    vtFrom DATE,
    vtTo DATE,
    tt TIMESTAMP,
    PRIMARY KEY (id,vtFrom)
);
with clustering order by (vtFrom DESC);
```

Theoretically, Cassandra cannot be classified as a temporal database, but it does provide strong capabilities to handle temporal data. Although Cassandra does not have interval (period) datatype but still can give us the ability to define our own time that has a start and end timestamps. One way is to add a column that contains interval that represents the valid time period the same way we deal with the timestamp. The interval will be of the form [validTimeS, validTimeE] and can be attached to the values in column families or to the columns. The data in columns can be modeled as a pair of (value, interval) for example (Programmer, [1, 4]). Table 3 shows how we could represent data in column oriented NoSQL databases.

Table 3: Embedding temporal data into Column Stores

ID	Name	Salary
1	Sami	(500,[1,2]),
		(700,[3,4]),
		(1000[5,∞])

```
CREATE TYPE salary (
        amount FLOAT,
        vtFrom TIMESTAMP,
        vtTo TIMESTAMP
    );
CREATE TABLE employee (
        id INT PRIMARY KEY,
        name TEXT,
        payroll      map<TIMESTAMP,
  frozen<salary>>
    );

INSERT INTO employee(id, name,
payroll)
  VALUES (1,
          'Sami',
          {'1/1/2015':
          {salary: 500,
            vtFrom: '1/1/2015',
            vtTo: '31/5/2016',
     }});

UPDATE employee
  SET  payroll = payroll
  +{'5/6/2016':
   { salary: 50000,
     vtFrom: '1/6/2016',
     vtTo: '1/10/2017' }}
   WHERE id = 1;
```

5. CONCLUSIONS

Handling temporal data and adding bitemporal characteristics to the non-relational databases is a hot research area. The need for this implementation is obvious because of the increasing popularity of NoSQL databases and the vast number of applications that require temporal data management. In this paper, we presented the three main categories of NoSQL databases and proposed set of solutions to describe how we can manage bi-temporal characteristics. We discussed three different solutions to store valid and transactions times in Key-value databases, and two solutions to handle the bitemporal properties in column oriented stores. In addition, we used Redis and Cassandra stores to show how we can embed bitemporal properties in the NoSQL databases. In the future work, we seek to embed the bitemporal properties in document and graph databases. Moreover, we are concerned about the temporal data operators such as before, after and union.

ACKNOWLEDGEMENTS

This work was supported in part by Zayed University Research Office, Research Cluster Award#17079.

6. REFERENCES

[1] M.D. Monger, R.A. Mata-Toledo, P. Gupta, "Temporal data management in NoSQL databases," *Journal of Information Systems & Operations Management*, vol. 6, issue 2, pp. 1-7, 2012.

[2] K. Grolinger, W.A. Higashino, A. Tiwari, M.A.M. Capretz, "Data management in cloud environments: NoSQL and NewSQL data stores," *Journal of Cloud Computing: Advances, Systems and Applications*, vol. 2, issue 1, article no. 49, December 2013.

[3] J. Bhogal, I. Choksi, "Handling big data using NoSQL," *Proceedings of the 2015 IEEE 29th International Conference on Advanced Information Networking and Applications Workshops (WAINA)*, 24-27 March 2015, pp. 393-398.

[4] V.N. Gudivada, D. Rao, V.V. Raghavan, "NoSQL systems for big data management," Proceedings of the 2014 IEEE World Congress on Services (SERVICES), 2014, pp. 190-197.

[5] K. Kaur, R. Rani, "Modeling and querying data in NoSQL databases," Proceedings of the 2013 IEEE International Conference on Big Data, 6-9 October 2013, pp. 1-7.

[6] F. Chang, J. Dean, S. Ghemawat et al., "Bigtable: A distributed storage system for structured data," *ACM Transactions on Computer Systems (TOCS)*, vol. 26, issue 2, article no. 4, 2008.

[7] G. DeCandia, et al., "Dynamo: amazon's highly available Key-value store," *ACM SIGOPS Operating Systems Review*, vol. 41, issue 6, pp. 205-220, 2007.

[8] G. Harrison, "Sharding, Amazon, and the Birth of NoSQL," in *Next Generation Databases*, Springer, pp. 39-51, 2015.

[9] P. Atzeni, F. Bugiotti, L. Rossi, "Uniform access to non-relational database systems: The SOS platform," *Proceedings of the International Conference on Advanced Information Systems Engineering*, 2012, pp. 160-174.

[10] N.Q. Mehmood, R. Culmone, L. Mostarda, "Modeling temporal aspects of sensor data for MongoDB NoSQL database," *Journal of Big Data*, vol. 4, issue 8, pp. 1-35, 2017.

[11] C.J. Tauro, S. Aravindh, A. Shreeharsha, "Comparative study of the new generation, agile, scalable, high performance NOSQL databases," *International Journal of Computer Applications*, vol. 48, issue 20, pp. 1-4, 2012.

[12] A. Moniruzzaman, S.A. Hossain, "Nosql database: New era of databases for big data analytics-classification, characteristics and comparison," arXiv preprint arXiv:1307.0191, 2013.

[13] J. Pokorny, "NoSQL databases: a step to database scalability in web environment," *International Journal of Web Information Systems*, vol. 9, issue 1, pp. 69-82, 2013.

[14] A. Auradkar, et al., "Data infrastructure at LinkedIn," Proceedings of the IEEE 28th International Conference on Data Engineering (ICDE'2012), 2012, pp. 1370-1381.

[15] Redis. [Online]. Available at: https://redis.io.

[16] Riak Products. [Online]. Available at: http://basho.com/products/.

[17] R. Hecht, S. Jablonski, "NoSQL evaluation: A use case oriented survey," Proceedings of the 2011 IEEE International Conference on Cloud and Service Computing (CSC), 12-14 December 2011, vol. 1, pp. 336-341.

[18] Apache HBase, [Online]. Available at: https://hbase.apache.org.

[19] G. Wang, J. Tang, "The nosql principles and basic application of cassandra model," *Proceedings of the IEEE 2012 International Conference on Computer Science & Service System (CSSS)*, 2012, pp. 1332-1335.

[20] Apache Accumulo, [Online]. Available at: https://accumulo.apache.org.

[21] K. Banker, *MongoDB in Action*, Manning Publications Co., 2011.

[22] J. C. Anderson, N. Slater, J. Lehnardt, *CouchDB: The Definitive Guide*, (1st ed.), O'Reilly Media, 2009, p. 300.

[23] A. Cuzzocrea, "Temporal aspects of big data management: state-of-the-art analysis and future research directions," Proceedings of the 2015 22nd IEEE International Symposium on Temporal Representation and Reasoning (TIME), 23-25 September 2015, pp. 180-185.

[24] A. Cuzzocrea, I.-Y. Song, "Big graph analytics: The state of the art and future research agenda," *Proceedings of the 17th ACM International Workshop on Data Warehousing and OLAP (DOLAP'14)*, 7 November 2014, pp. 99-101.

[25] F. Bugiotti, et al., "Database design for NoSQL systems," *Proceedings of the International*

Conference on Conceptual Modeling, 2014, pp. 223-231.

[26] Y. Hu, S. Dessloch, "Defining temporal operators for column oriented NoSQL databases," *Proceedings of the East European Conference on Advances in Databases and Information Systems (ADBIS'2014)*, 2014, pp. 39-55.

[27] MarkLogic, *Best Database For Integrating Data From Silos*. [Online]. Available at: www.marklogic.com.

[28] J. Han, et al., "Survey on NoSQL database," *Proceedings of the 6th International Conference on Pervasive Computing and Applications (ICPCA'2011)*, 26-28 October 2011, pp. 363-366.

Mohammed Eshtay has received his PhD in Computer Science from university of Jordan in 2018. He has many years of experience in the field of Information Technology working in various positions. In addition to his work in the academic field, he has held many positions in the industry. Dr. Eshtay research interests include machine learning, evolutionary computing, data science and NoSQL stores.

Azzam Sleit is the Former Minister of ICT from 2013 to 2015. He is a Professor of Computer Science with the University of Jordan, where he also functioned as the Dean of IT& Director of the Computer Center. Dr. Sleit has over twenty five years of experience and leadership in the IT field working at various levels of government, private and international sectors. Dr. Sleit is the author of more than one hundred research papers published in reputable journals and conferences.

Monther Aldwairi is an associate professor at the College of Technological Innovation at Zayed University since fall of 2014. He received his B.S. in electrical engineering from Jordan University of Science and University (JUST) in 1998, and his M.S. and PhD in computer engineering from North Carolina State University (NCSU), Raleigh, NC, in 2001 and 2006, respectively. Prior to joining ZU, he was an Assistant and then Associate Professor of Computer Engineering at Jordan. University of Science and Technology. He served as the Vice Dean of the Faculty of Computer and Information Technology from 2010 to 2012 and was the Assistant Dean for Student Affairs in 2008. In addition, he was an Adjunct Professor at New York In- stitute of Technology (NYiT) from 2009 to 2012. He worked at NCSU as Post-Doctoral Research Associate in 2007 and as a research assistant from 2001 to 2006. He worked as a system integration engineer for ARAMEX from 1998 to 2000. Dr. Aldwairi's research interests are in information, network and web security, intrusion detection, dig- ital forensics, reconfigurable architectures, parallel architectures and algorithms, artificial intelligence and pattern matching algorithms.