International Journal of Computer (IJC)

ISSN 2307-4523 (Print & Online)

http://ijcjournal.org/

To have an Idea on NoSQL Databases

Bruno Sadeg[a*], Claude Duvallet[b]

[a,b]*LITIS Laboratory - University of Le Havre Normandy*

[a]*Email: bruno.sadeg@univ-lehavre.fr*

[b]*Email: claude.duvallet@univ-lehavre.fr*

Abstract

NoSQL databases (initially non-SQL, then Not Only SQL) are specifically designed to handle large amounts of data. They have been developed since the 1970s, but they have gained the interest of academia and industry for about two decades. This is because of their powerful characteristics and lack of relational databases, which are the most widely used data sources around the world. Indeed, these databases are based on the relational model, which is materialized by a relational database management system (RDBMS). Although RDBMS efficiently manage data (tables), they have many drawbacks that make them unsuitable for managing current data, which come mainly from Internet applications. They are called Big Data and they are used for example by Twitter, FaceBook, LinkedIn, ... They are very numerous and tend to change quickly. In fact, among the disadvantages of relational databases, we can mention: non-flexibility, non-scalability, ... On the contrary, NoSQL databases evolve very well (scaling) and almost all NoSQL databases are schema-free (we can add or delete an entity or a relationship at any time during execution). In this article, we begin by giving an overview of relational databases and their characteristics. We then describe the NoSQL databases and their main characteristics, knowing that there are as many different characteristics as "NoSQL databases" products. We then give the taxonomy of NoSQL databases, which distinguishes four main types of NoSQL databases: key-value, wide-column, document and graphical databases. We will then give some elements of each type of database through the use of a product, an implementation of a kind of such a database.

Keywords: Relational model ; NoSQL database ; scalability ; schemaless ; BigData.

---

* Corresponding author.

1. Introduction

Since the 1970's, the relational model [1] has been widely used in academic and industry areas. This is due to its simplicity and its theoretical foundations. Indeed, the model is based on robust mathematical theory (relational and sets theory) [2]. The first materialization of the relational model was the development of DB2 database [3] and DB2 relational database management system (RDBMS). Despite the performance issues encountered by relational DBMS at the beginning of their implementation, many systems have been developed since then. This is due to the very hard work on different optimizations of the RDBMSs made by a wide community of databases scientists. The consequence is the powerful of the existing RDBMSs due to different optimizations. So, RDBMSs have become unavoidable in any business and academic areas. However, these last twenty last years, new applications arisen with the development of the WEB network. These new applications deal with great quantities of heterogeneous data and sometimes, connected data. RDBMSs quickly became unsuitable for managing this data (called Big Data) [4]. Then a new concept of database raised: NoSQL databases [5]. Most of these databases are schemaless and are very scalable, and most of them do not follow the ACID behavior, instead, they obey to BASE [6] principle (have properties which relax consistency property). The only kind of NoSQL database whose transactions have natively ACID properties is graph databases. In this paper, we recall the characteristics of the widely used DBMS: the relational model [7]. We highlight its advantages and its drawbacks. We then present briefly NoSQL databases and their taxonomy. We describe the different kinds of such systems. We illustrate each kind of NoSQL database [8, 9] by using a product, representative of this kind of database: we illustrate key-value database by REDIS [10], wide column database by CASSANDRA[11], document database by MongoDB [12] and a graph database by Neo4j [13]. We conclude the paper by summarizing its contents and by giving a few research directions we are planning to follow.

2. SQL vs NoSQL databases

2.1. SQL databases (Relational databases)

SQL databases (more precisely, relational databases) [14] are databases where data are structured in the form of tables (in almost all tables, rows represent entities, whereas columns represent the characteristics of these entities); These tables are called *base tables*. There exists an other kind of tables which serve as a link between base tables. They contain keys, called foreign keys, which reference primary keys of base tables. To retrieve the complete information about two or more entities, one have to use the join operations between these tables.

2.1.1. Illustrative example

Assume we want to model a schooling system. We have two base tables (entities): Teacher and Course, with their characteristics. Another table (Teach table) is used to link these two base tables.

Each table has its own characteristics. Among these characteristics, we find a primary key of each table:

- num_t: integer, the primary key of Teacher table,

- cod_c: integer, the primary key of Course table,

- (num_t, cod_c): primary key of Teach table. It is composed of the two attributes num_t and cod_c. In the same time num_t and cod_c are the foreign keys of teach table, which reference respectively Teacher table and Course table.

Example of request: find all courses teached by a teacher whose name is 'ARY'.

SQL query [14] :

```
SELECT name_s FROM Teacher, Teach, Course
WHERE  teacher.num_t = Teach.num_t
  AND    Teach.cod_c = Course.cod_c
  AND     Teacher.name_t = 'ARY';
```

Table 1: Teacher table

num_t	name_t	adr_t	phone_t
1	'ARY'	'adr_t1'	'phone_t1'
2	'DAT'	'adr_t2'	'phone_t2'

Table 2: Course table

cod_c	name_c	coeff_c
'c1'	'GEO'	4
'c2'	'MATH'	5

Table 3: Teach table

num_t	cod_c	nb_hours
1	'c1'	10
1	'c2'	15
2	'c2'	25

Teacher X Teach (Cartesian product): Result table

Table 4

1	‘ARY’	‘adr_t1’	‘phone_t1’	**1**	‘c1’	10
1	‘ARY’	‘adr_t1’	‘phone_t1’	**1**	‘c2’	15
1	‘ARY’	‘adr_t1’	‘phone_t1’	2	‘c2’	25
2	‘DAT’	‘adr_t2’	‘phone_t2’	1	‘c1’	10
2	‘DAT’	‘adr_t2’	‘phone_t2’	1	‘c2’	15
2	‘DAT’	‘adr_t2’	‘phone_t2’	**2**	‘c2’	25

Intermediate result (teacher.num_t = Teach.num_t):

Table 5

1	‘ARY’	‘adr_t1’	‘phone_t1’	**1**	‘c1’	10
1	‘ARY’	‘adr_t1’	‘phone_t1’	**1**	‘c2’	15
2	‘DAT’	‘adr_t2’	‘phone_t2’	**2**	‘c2’	25

(Teacher X Teach) X Course (Teach.cod_c = Course.cod_c):

Table 6

1	‘ARY’	‘adr_t1’	‘phone_t1’	**1**	**‘c1’**	10	**‘c1’**	‘GEO’	4
1	‘ARY’	‘adr_t1’	‘phone_t1’	**1**	‘c2’	15	‘c1’	‘GEO’	4
2	‘DAT’	‘adr_t2’	‘phone_t2’	**2**	‘c2’	25	‘c1’	‘GEO’	4
1	‘ARY’	‘adr_t1’	‘phone_t1’	**1**	‘c1’	10	‘c2’	‘MATH’	5
1	‘ARY’	‘adr_t1’	‘phone_t1’	**1**	**‘c2’**	15	**‘c2’**	‘MATH’	5
2	‘DAT’	‘adr_t2’	‘phone_t2’	**2**	**‘c2’**	25	**‘c2’**	‘MATH’	5

Final result (projection on the asked attributes) (i.e. cod_c and name_s):

Table 7

cod_c	**name_s**
‘c1’	‘GEO’
‘c2’	‘MATH’
‘c2’	‘MATH’

Note : with DISTINCT clause in SQL query, we will not obtain the duplicate rows.

2.1.2. Comments

With a relational database [7], if we want to make a link between two or more tables (here Teacher and Course tables), we have to join these tables. If we have to link *n* tables, we have to do *n* join operations. But, it is well known that the join operation is very costly. Then if we have to work with connected data, relational databases are not suitable. This is one of the drawbacks of the relational model. This will not happen with NoSQL databases (see for example the same example with MongoDB database [16] in section 2.2.3.2.). We will see in the rest of this paper other drawbacks of this model.

2.1.3. Relational DataBase Management System (RDBMS)

RDBMS is a software (or a set of softwares) developed to manage relational databases [1]. Its main component is the SQL language (Structured Query Language). SQL language is a standard used by all products implementing relational database like Oracle [14], PostgreSQL, MySQL and so on. It is composed of three sets of commands and some other useful commands used to manage relational databases, which are:

- Data Description Language (DDL): used to describe, remove, alter or drop the database objects (tables, views, indexes, columns, ...)
- Data Manipulation Language (DML): used to insert, update or delete data (into/from) the database objects (tables, tablespaces, indexes, columns, ...)
- Data Control Language (DCL): used to manage the authorizations to grant (or deny) accesses to (from) a database. The DCL sub-language is also used to control the accesses (CREATE, ALTER, DROP, INSERT, …) to the database objects: tables, views, columns, ...

There are two other main commands: COMMIT (to commit a transaction) and ROLLBACK (to rollback a transaction). A transaction in a relational database has ACID properties, which mean:

- Atomicity: means that all the actions (read/write) composing the transaction commit. If at least one action fails, the transaction will be aborted: “ALL OR NOTHING” principle,
- Consistency: means that if the transaction is well-defined, when we apply it on a consistency database, the database remains consistent,
- Isolation: means that when a transaction execute, its updates are not “seen” by other transactions before it commits,
- Durability: means that when a transaction commits, its effects become persistent in the database, even a failure occurs.

Note: as we’ll see later in the paper, almost all NoSQL databases [17] do not follow ACID properties as these properties are too strict. Thus, these databases relax notably the consistency property in order to gain in performance.

2.2. NoSQL databases

With the development of Internet applications, large amounts of heterogeneous data are produced. These data can be text, images, videos, photos, ... To manage these data, relational databases proved their powerlessness due mainly to their non-flexibility and to non-scalability. To overcome these problems and to design efficient systems that allow flexibility in the scheme and ease of scaling (among other properties), a new kind of databases has been raised. They are called under the term of "NoSQL" [18] (originally meaning No SQL, then Not Only SQL) databases [5], but in fact this means “not relational” databases. Some of them have an SQL-like language to manipulate data, some others don’t. Here are listed some main characteristics of NoSQL databases:

- Flexibility: since they are schemaless systems, one can add or remove entities and relationships to the database, easily.
- Scalability: NoSQL databases deal with billions of entities by scaling horizontally (adding other servers), unlike scaling vertically, where one must power a server by adding components (Memory, Processors, …).
- Fault-tolerance; NoSQL databases are generally distributed over multiple nodes. Then, if one node fails, its data become unavailable, these data may be obtained from other nodes thanks to replication policy.
- Availability: still thanks to data replication on many nodes, data are always available.

NoSQL databases deal more efficiently than relational databases with Big Data, which are huge amounts of heterogeneous data. This data have commonly the so-called 3 V quality (Volume, Variety and Velocity) where Volume indicates that there are trillions data (for example, Facebook stores about 200 billions images-photographs), Variety is related to different kinds of data (images, texts, multimedia, …) and Velocity means that data change very quickly over time. Besides, NoSQL databases follow the CAP [19] theorem (Consistency, Availability and Partition) where Consistency means that data in the database are consistent, Availability means that data is always available (notably thanks to replication), Partition means that the database is tolerant to the partition of data. CAP’s theorem [19] says that whatever a database, it follows at most two of the CAP properties. Almost NoSQL databases have BASE [6] properties (not ACID): these terms are originated from chemical vocabulary. ACID, BASE. The definition of ACID acronym is given in Section 2.1.3, whereas BASE properties stand to BAsic availability, Soft state, and Eventual consistency where Basic availability: means that, in NoSQL databases, data is always available thanks to replication (if a failure occurs in a site, data will be available on other replicated sites), Soft state: means that the database may not be consistent all the time, Eventual consistency: means that updates on the database will eventually be applied through to all servers, not immediately but given enough time. Generally, we distinguish four types of NoSQL Databases: key-value [20], wide column [21], document [20] and graph databases [22, 23]. In the following sections and sub-sections, we describe the characteristics of each kind of database and use an example of each kind to illustrate the database.

2.2.1. Key-Value databases

2.2.1.1. Characteristics

A key-value database [20] (key-value store or key-value store database) is a kind of NoSQL database that uses a simple key/value method to store data. It refers to the fact that such a database stores data as a collection of key/value pairs. Its main characteristics is (I) its simplicity, (ii) the fact that it scales very well and (iii) it has great performances as it accesses data very fastly. The key in a key-value pair must be unique. It allows the access to the value associated with this key. The key value may depend on the DBMS used which can impose limitations. The value in a key-value pair can be anything, such as text, string,, short int, number, markup code (HTML, PHP, …), an image, …

2.2.1.2. Some use cases and products

Key-value databases can be used efficiently in many areas: General Web/Computers (User profiles, Session information, Blog comments, Emails), E-commerce (Shopping cart contents, Product categories, Product details, …), Networking and Data Maintenance (Telecom directories, Internet Protocol (IP), …).

There are several key-value databases. Some of them are: REDiS, Voldemorte, AeroSpike, …

2.2.1.3. An illustration of a key-value database: REDIS

ReDiS (Remote Dictionary Server) [10] is an-memory data project which implements key-value database. It uses various types of data structures: bitmap, maps, strings, lists, sets, streams, and spatial indexes, ...

Some commands of REDIS:

> SET toto "Hello World"

OK // setting the key toto to the string "Hello World"

> GET toto

"Hello World" // *getting the value of the key toto*

> DEL toto

(integer) 1 // *deleting the key toto*

> GET toto

(nil) // *key toto has no value*

> SETEX toto 40 "the sentence is: Hello World!"

OK // Setting the key toto. it has 40 seconds before expiration

> TTL toto

(integer) 15 // *15 secods before timeout*

> PERSIST toto

(integer) 1 *// making persistent the key toto*

> RENAME toto toto1

OK // *renaming the key toto as toto1*

Note: If we implement the schooling example with REDIS database, we'll have keys and values: it's possible to consider each teacher name as a key. The associated value to a teacher name may be a set of courses teached by this teacher. Then, if we want to find all courses teached by a giving teacher, we give his name and we'll obtain all courses he teaches.

2.2.2. Wide-Column databases

2.2.2.1. Characteristics

Wide column databases [24] are a kind of NoSQL databases that work very well for storing great amounts of data that can be collected. They allow to manage data that won't fit on one computer. Wide column are database management systems that organize related facts into columns. These columns form groups ("column families"). They have content and function similar to tables in relational databases. Wide column stores are suitable in distributed systems (when there are much data to distribute in many computers). *Wide column databases use columns and rows to store data, but unlike relational databases, the rows may not have the same number of columns. They store data as row-values, column-values and timestamps.*

2.2.2.2. Some use cases, products and an example of wide column value database: CASSANDRA

Wide column databases are used in many ares. Among them: IoT, *User preferences, Geographic information, Time Series Data, Logging applications and so on.*

Among the variety of wide-column databases, we can cite: Cassandra, HBase, Accumulo, Google's BigTable, …

CASSANDRA database:

Cassandra [25] is a NoSQL distributed database which allows to store a great quantities of data, notably thanks to its horizontal scalability. It belongs to wide column NoSQL databases. In addition, it is schemaless and fault-tolerant database. Its data model is issued from Google Big Table and its distribution design is based on Dynamo Amazon). In distributed Cassandra system, all nodes play the same role: there is no master, nor slaves. So, there is no single point of failure. Originally, the CASSANDRA database was developed by Facebook to manage its

messaging: the database had not many functionalities. Since late 2000's, CASSANDRA has been managed by Apache company who add some following characteristics:

- High availability: CASSANDRA allows the user to relax consistency (more precisely, it's possible for the user to define which level of consistency he wants for reads and for writes). We speak about Tuning Consistency.
- Extended data model: it's possible for the user to add or remove entities or relationships on the fly (during the utilization of the database). The database is schemaless.
- Fault-tolerance: data localized on a node are automatically replicated on other nodes. Thus, if some data are unavailable on one node because it's faulty, these data may be obtained from an other node,
- Decentralized control: since in CASSANDRA, there is no master node, nor slave node, no node constitutes a bottleneck for data.

CASANDRA concepts:

- Column: is the littlest concept of CASSANDRA. It's a triplet composed of a name, a value and a timestamp. This later indicats the instant of the value last update.
- Row: is composed of a set of columns and is identified by a key. A column name may be used as a key. Each row may have columns with different lengths.
- Column family: is a logical set of rows (equivalent to a relational table)
- Keyspace: is is a family group of columns (equivalent to a schema in relational model)..

There exist two command tools to manipulate a database wih CASSANDRA: Cassandra-CLI and CQLSH (Cassandra Query Language), based on Python language.

Here are some commands of Cassandra:

- Show keyspaces:

CASSADRA-CLI: > SHOW keyspaces;

CQLSH: > SELECT * FROM system.schema_keyspaces;

- Create a keyspace: (is equivalent to a database in relational model):

> create keyspace kspace_name with replication_factor= 2

- Using this keyspace:

The two tools: > use keyspace_name

- Create a column family Person (to write values into columns):

CQLSH:

> USE keyspace_name;

> INSERT INTO Person (firstName, lastName, address, phone) VALUES ('Jean', 'TUCHEL', 'Rouen', '+33617655400');

Cassandra-CLI:

> USE keyspace_name;

> SET Person['jtuchel']['firstName']='Jean';

> SET Person['jtuchel']['familyName']='TUCHEL';

> SET Person['jtuchel']['address']='Rouen';

> SET Person['jtuchel]['phone']='+33617655400';

- Finding how many columns we have:

> count keyspace_name ['marc']

- Reading the columns:

Cassandra-CLI:

> get Paerson['jtuchel']

=> (column=666e616d65, value=jean, timestamp=1182590343020)

=> (column=656d61696c, value=TUCHEL, timestamp=1182313429021)

=> (column=666e616d65, value=Rouen, timestamp=1182590344321)

=> (column=656d61696c, value=+33617655400, timestamp=1182313443231)

Returned 4 results.

- Deleting phone of 'jtuchel':

> del keyspace_name['jtuchel']['phone']

- Deletineg all the 'jtuchel' row:

Cassandra-CLI:

> del keyspace_name['jtuchel']

CQLSH:

> DELETE FROM Person WHERE lastName='TUCHEL';

UPTATES:

Cassandra CLI:

> UPDATE KEYSPACE keyspace_name WITH strategy_options = {replication_factor:2};

CQLSH:

> ALTER KEYSPACE keyspace_name WITH strategy_class=SimpleStrategy AND strategy_options:replication_factor=1;

DROPPING a keyspace:

With the two tools: > DROP KEYSPACE keyspace_name;

2.2.3. Documents databases

2.2.3.1. Characteristics

A document-oriented database [20] , is a kind of NoSQL database where the main entity is a document. Each document has a unique key (identifier) and the documents have not the same length. For example, we can have one document with five names, associated addresses and five ages, and we can have an other document with three names and three phone numbers. Document database will store data of both documents whatever their types and it is possible for it to handle variable length data sets. Consequently, document databases have flexible schema; they are schemaless (documents can have different schema and data values, unlike tables in a relational model where each row is related to an entity and will have same columns structure). Document databases use various encodings and formats to store data (XML, JSON, BSON, …).

2.2.3.2. Use case and an example of document database: MongoDB

We consider the schooling example depicted in section 2.1.1.

Schooling system example processed with MongoDB document database [26, 27] :

we will have the following document:

{ “name_t “: “ARY”,

“adr_t “: “adr_t1”,

“phone_t “: “phone_t1”,

“Course:“

[

“name_c:“ “name_c1”, “coef_c1“: 4, “nb_hours“: 10,

“name_c“: “name_c2”, “coef_c2“: 5, “nb_hours“: 15

]

“name_t “: “DAT”,

“adr_t “: “adr_t2”,

“phone_t “: “phone_t2”,

“Course“:

[

“name_c“: “name_c2”, “coef_c2“: 5, “nb_hours“: 25

] }

Note: if we want to retrieve the course teached by a teacher whose name is “ARY”, we have no join operation to do, because we have all the information in the same document !

db.teachers.find ({name_t: "ARY"})

2.2.4. Graph databases

Graph databases [28, 29, 30] are a kind of NoSQL databases. They are efficiently used to manage great quantities of connected data [31]. These data may be structured, semi-structured or unstructured.

2.2.4.1. Characteristics

Designing a graph database is very natural: the database model is the same as the physical model. Here is an example where we have three persons (Eric, Marc and Jacques) connected by two relationships (is_friend_of,

teach_to) :

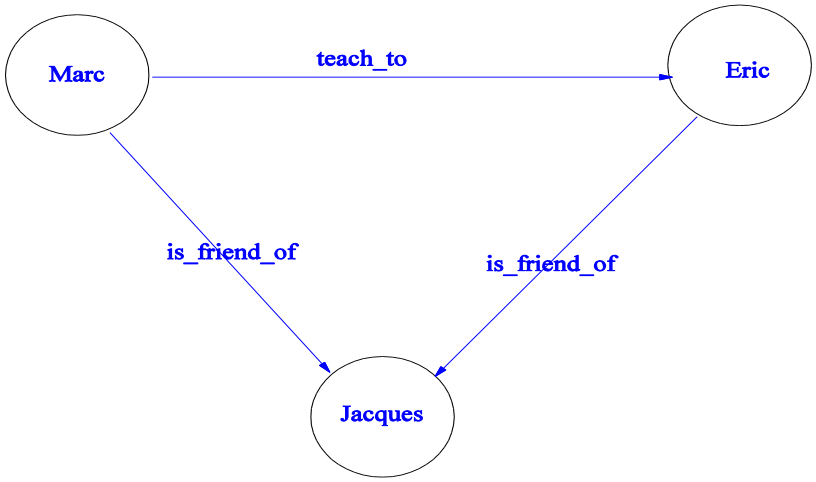

Figure 1

2.2.4.2. Some use cases and products

Nowadays, many WEB applications deal with connected heterogeneous data. Examples of such applications are Facebook, Twitter, Network & IT Operations, Fraud Detection, Real-time Recommendations, Access Management, an so on.

GraphDB, Neo4j are examples of graph databases. However, Neo4j is the most popular and robust graph database in the word.

2.2.4.3. Illustration of a graph database: Neo4j

2.2.4.3.1. What is Neo4j ?

Neo4j [32], written in Java and Scala, is an open-source NoSQL graph database [33, 34]. It has transactions which are compliant to ACID-properties. Neo4j database began to be developed in 2003 and has been available to the public since 2007. Neo4j is declined into two versions: Community Edition and Enterprise Edition. In addition to the characteristics of the Community Edition, the Enterprise Edition has specific business needs such as backups, clustering, and so on.

The following figure illustrates the main elements manipulated by Neo4j graph database.

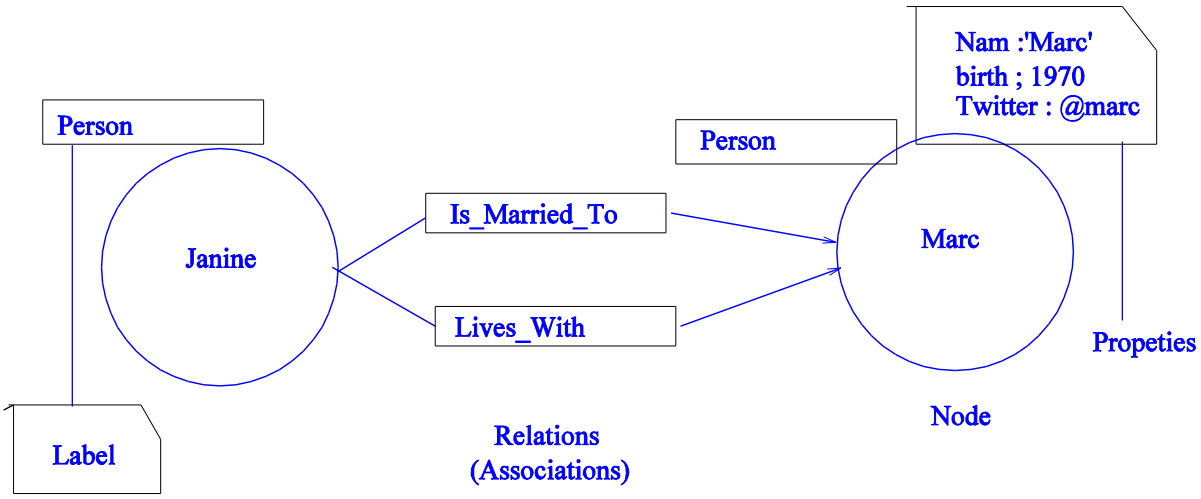

Figure 2

In the previous figure, we have two nodes labeled Person, whose values are Janine and Marc. Marc node has some properties name, birth year and Twitter address. The nodes are linked by two directed relationships: Janine IS_MARRIED_TO Marc, and Janine LIVES_WITH Marc.

Here are some characteristics of the Neo4j database which make it very popular and widely chosen by many enterprises:

- Flexibility: the property graph model on which is based Neo4j can adapt in a lot of situations over time. So, it is possible to add nodes and/or relationships to (or remove them from) an existing database later according to the enterprise needs. In other words, Neo4j is schemaless, say it has not a predefined schema.
- Cypher: To manipulate the database, Neo4j uses Cypher, a declarative query language similar to SQL for relational databases. Cypher is optimized for graphs. It functions by matchings and it has some commands similar to that of SQL (WHERE, ORDER BY, LIMIT, …).
- Constant response time: when we have a database (big graph), the response time to queries is almost the same, whatever the query (depth in the graph) due to efficient representation of nodes and relationships.
- Scalability: Neo4j is able to deal with great quantities of data (billions of nodes and relationships).
- Availability of drivers: Neo4j has drivers for several programming languages, including Java, JavaScript, .NET, Python, and many more.

2.2.4.3.2. Cypher language [35]

We'll illustrate the Cypher functioning through an example:

- Creation of nodes and relationships:

CREATE (n:Person:Teacher {name: "Marc", phone:"0634243220", born: 1973}) RETURN n

```
CREATE (n:Person:Student {name: "Eric",  phone:"0622415300", born: 1965}) RETURN n

MATCH (p1:Person), (p2:Person) CREATE (p1)-[r:Teach]-> (p2) return p1, r, p2
```

- Queries:

- Find the name and the year of birth of the all persons:

```
MATCH (p:Person) return distinct p.name, p.born
```

- Find the phone number the person whose name is Marc:

```
MATCH (p {name: "Marc"}) return distinct p.phone
```

- Find the student who is teached by the person named Marc:

```
MATCH (Marc)-[:Teach]->(p2:Person:Student)  return distinct p2.name, p2.born
```

4. About NoSQL databases

To deal with the increasing number of Internet applications which manipulate huge volumes of heterogeneous data, the emerging NoSQL technology [5] (declined into four main categories) seems the key issue. However, some researchers do not think so, although they agree that NoSQL databases have many characteristics very useful for the current Internet applications like scalability, performance and flexibility [18]. Indeed, these researchrs think that database consistency if very important in an application, but NoSQL databases do not provide this feature because almost of them have BASE properties (not ACID). So, these systems just assure eventual consistency of the database, which is not sufficient. Up to now, the debate remains open. Research is conducted that attempts to address the features that are missing from NoSQL databases, namely strict consistency.

5. Research issues

As we have said in the previous sections, NoSQL databases are power tools to manage current WEB applications which deal with great quantities of data. The main issue for researchers remains to maintain strict consistency of the database, while facilitating is scaling and flexibility in their schema. An other issue is to develop an SQL-like language for manipulating the database. This language must be standardized so as each product will use the same kernel language and extend it with its own characteristics.

6. Conclusion

In this paper, we began to give an overview on relational databases and on systems which manage them, illustrating them through an example. Then we gave some elements on NoSQL databases. After that, we descried each kind of NoSQL database representing each family of such databases. In the same time, for each

kind of NoSQL database, we illustrated it by using a few commands of a product implementing it. We then gave some open research issues on NoSQL databases.

References

[1] D. Isaac et al. "Hierarchical storage management for relational databases", Proceedings of Twenth IEEE Symposium on Mass Storage Systems, 1993, Monterey, CA, USA, USA, DOI: 10.1109/MASS.1993.289767.

[2] J.C. Date. Database Design and Relational Theory. Patparganj, Dehli; Publisher: Pearson Education Asia, 7th edition (2002).

[3] DB2. IBM DB2 Universal Database Administration Guide, Design and Implementation. (SC09-2839),

IBM Corp, 1999.

[4] Bansari H. Kotecha and H. Joshiyara. "A Survey of Non-Relational Databases with Big Data". International Journal on Recent and Innovation Trends in Computing and Communication (IJRITCC), vol. 5, issue 11, pp. 143-148, Nov. 2017.

[5] M. Madison, M. Barnhill, C. Napier and J. Godin. "NoSQL Database Technologies". Journal of International Technology and Information Management, vol. 24, issue 1, 2015, Available:

https://core.ac.uk/download/pdf/55333675.pdf

[6] Dan Pritchett, Ebay. "Base: An ACID Alternative". In ACMQUEUE, Vol. 6, Issue 3, Jul. 2008.

[7] J.C. Data. An Introduction to Database Systems. Addison-wesley, 2004.

[8] Adity Gupta, Swati Tyagi , Nupur Panwar, Shelly Sachdeva, Upaang Saxena, "NoSQL databases: Critical analysis and comparison" International Conference on Computing and Communication Technologies for Smart Nation (IC3TSN), Gurgaon, India, 2017.

[9] A. G. Mhmoud Mohmmed and S.E. Fatoh Osman. "Study on SQL vs. NoSQL vs. NewSQL". International Journal of Multidisciplinary Engineering Science Studies (JMESS), vol. 3, issue 6, pp. 1821-1823, Jun. 2017.

[10] T. Macedo and F. Oliveira. Redis CookBook. O'Reily, https://books.phundrak.fr/download/1948/pdf/1948.pdf, [2011].

[11] P. Shukla. "NoSQL Database: Cassandra is a Better Option to Handle Big Data". International Journal of Science and Research (IJSR), vol. 5, num. 1, pp. 24-26, Jan. 2016.

[12] Tutorials point. “Mongo DB tutorial”. Available: mdslab.unime.it/sites/default/files/mongodb_tutorial.pdf [2003].

[13] A. Turu Pi and O. Koroglu. “Graph Databases and Neo4j.” PhD thesis, Université Libre de Bruxelles https://cs.ulb.ac.be/public/_media/teaching/neo4jj_2017.pdf [2017].

[14] D. Florescu. “ Oracle”, in the “ACM Queue”, Vol. 3, No.8, 2005, Available: dist.neo4j.org/neo-technology-introduction.pdf.

[15] A. Nayak, A. Poriya and D. Poojary. “Type of NOSQL Databases and its Comparison with Relational Databases”. International Journal of Applied Information Systems (IJAIS), vol. 5, issue 4, pp. 16-19, Mar. 2013.

[16] K. Seguin. The Little MongoDB Book. Available: https://openmymind.net/mongodb.pdf [Mar 2011].

[17] Behjat U Nisa. “A Comparison between Relational Databases and NoSQL Databases”. International Journal of Trend in Scientific Research and Development (IJTSRD), vol. 2, issue 3, pp. 845-848, Mar.-Apr. 2018.

bytes.usc.edu/cs585/s19_data0AI2AGI4/extras/docs/Graph_Dbs_for_Beginners.pdf[2018]..

[18] M. Stonebraker. “SQL databases v. NoSQL databases”, Communications of the ACM, vol. 53, issue. 4,

pp. 10-11, [2010].

[19] S. Gilbert and N. Lynch. “Perspectives on the CAP Theorem”. In IEEE Computer, Vol. 45 , Issue: 2, pp. 30-36, Feb. 2012.

[20] N. Dasharath Karande. “A Survey Paper on NoSQL Databases: Key-Value Data Stores and Document Stores”. International Journal of Research in Advent Technology, vol. 6, num. 2, pp. 153-157, Feb. 2018.

[21] G. Matel. “Column-Oriented Databases: an Alternative for Analytical Environment”. Database Systems Journal, vol. I, num. 2, pp. 3-15, 2010.

[22] S. Patil, G. Vaswani and A. Bhatia. “Graph Databases- An Overview”. International Journal of Computer Science and Information Technologies (IJCSIT), vol. 5, issue 1, pp. 657-660, 2014.

[23] X. Yan, P.S. Yu, J. Han. “Substructure Similarity Search in Graph Databases,” in Proc. of the ACM SIGMOD, Baltimore, Maryland, USA, Jun. 2005.

[24] D. Abadi, P. Boncz, S. Harizopoulos, S. Idreos and S. Madden. “The Design and Implementation of Modern Column-Oriented Database Systems”. Fondations and Trends in Databases Journal, vol. 5, num. 3, pp. 197-280, 2012.

[25] E. Hewitt. Cassandra: The Definitive Guide. O'reily Ed., e-book, Available:

https://www.gocit.vn/files/Cassandra.The.Definitive.Guide-www.gocit.vn.pdf [2011].

[26] K. Banker. "MongoDB in Action," Available: img105.job1001.com/upload/adminnew/2015-04-07/1428394945-PHQK1Q5.pdf [Apr 2015].

[27] Kristina Chodorow. MongoDB: The Definitive Guide. Second Edition by (O'Reilly). Copyright 2013 Kristina Chodorow, 978-1-449-34468-9. https://www.gocit.vn/files/MongoDB-www.gocit.vn.pdf

[28] S. Srinivasa. "Data, Storage and Index Models for Graph Databases". International Institute of Information Technology, Bangalore, aug. 2011, ISBN = 9781613500538.

[29] J. Pokorný. "Graph Databases: Their Power and Limitations", Proceedings of the 14th Computer Information Systems and Industrial Management (CISIM) Warsaw, Poland. pp.58-69. 2015.

[30] B. Merkl Sasaki, J. Chao and R. Howard. Graph Databases for Beginners. Neo4j: the #1 Platform for Connected Data. E-book, https://neo4j.com/lp/book-graph-databases/

[31] I. Robinson, J. Webber and E. Eifrem. Graph Databases: New Opportunities for Connected Data. Ed. Oreily,Second Edition, 2015. [On-line]. Available:

https://pdfs.semanticscholar.org/f511/7084ca43e888fb3e17ab0f0e684cced0f8fd.pdf

[32] F. Melchor Santos López and E. Guillermo Santos De La Cruz. "Literature review about Neo4j graph database as a feasible alternative for replacing RDBMS". Systemas E Informatica, vol. 18, issue 2, pp. 135-139, 2015.

[33] S. Choudhury., L. Holder, G. Chin, P. Mackey, K. Agarwal. and J. Feo. "Query Optimization for Dynamic Graphs," in Proc. of ACM, jul. 2014.

[34] S. Batra and C. Tyagi. "Comparative Analysis of Relational And Graph Databases". International Journal of Soft Computing and Engineering (IJSCE), vol. 2, issue 2, pp. 509-512, 2012.

[35] N. Francis et al. "Cypher: An Evolving Query Language for Property Graphs," In ACM SIGMOD, 2018.