Review paper
https://doi.org/10.31784/zvr.7.1.9
Received: 14 January 2019
Accepted: 25 February 2019

QUERYING DATA IN NOSQL DATABASES

Andrea Babić
Univ. Bacc. Inf., student, University of Rijeka, Department of Informatics, Radmile Matejčić 2, 51 000 Rijeka, Croatia; e-mail: andreababic051@gmail.com

Danijela Jakšić
PhD, Postdoctoral Researcher, University of Rijeka, Department of Informatics, Radmile Matejčić 2, 51 000 Rijeka, Croatia; e-mail: danijela.jaksic@inf.uniri.hr

Patrizia Poščić
PhD, Full Professor, University of Rijeka, Department of Informatics, Radmile Matejčić 2, 51 000 Rijeka, Croatia; e-mail: patrizia@inf.uniri.hr

ABSTRACT

The goal of this paper is to give an overview of fundamental concepts and types of NoSQL databases, to show some examples of database queries, some related research, and the implementation of those queries in an original practical example. The introduction is a brief representation and description of the NoSQL database. There are also several comparisons of NoSQL database with the relational database. The next chapter contains a review of the basic NoSQL databases and their prototypes. In each of the following subchapters, the types of NoSQL databases are described in more detail and various queries which can be performed over them are presented. In the last chapter there is also a practical example of querying one of these databases.

Key words: *NoSQL, database, key – value, document, queries*

1. INTRODUCTION

With increased demands for greater flexibility, better storage performance, and processing the large amounts of data, the need for a new database type has emerged over the last few years. Such databases are called NoSQL (Not Only SQL) databases and are not based on the relational data model. Unlike relational databases, NoSQL databases do not use SQL query language but instead they have their alternate languages. In addition, they are dynamic and do not require a data schema definition, which means that it is not necessary to define the structure beforehand. For this reason, the type and number of attributes of an entity can be changed without interruption. Another important feature is the distribution of NoSQL databases, that is, the ability to work on a larger number of computers, resulting in better performance, greater scalability, and work with much larger-scale databases in less time. On the other hand, relational databases are designed to work just on a single computer, meaning that data processing takes more time. (Marić, 2015; Stojanović, 2016).

The NoSQL movement has begun due to the increase in traffic on the web pages and the need to store more and more data. Relational databases did not represent an ideal solution and therefore, Google and Amazon were among the first to turn to alternative, non-relational databases that were more suited to specific market demands. Google has started using Bigtable database and Amazon Dynamo database. The conclusion is (Marić, 2015; McCreary, 2014) that there are four basic attributes that are responsible for the NoSQL movement:

- Volume – the amount of data that requires processing and storage in the database
- Speed – the need to process data as soon as possible, regardless of their amount
- Variability – a rigid scheme and lack of dynamics in relational databases pose a problem for changes that are therefore expensive and a lot of time is lost
- Agility – the need for simplicity when entering and retrieving data from the database (simpler queries)

1. 1 Data models

Since this review paper is mainly about querying data in NoSQL databases, NoSQL data models are out of scope of this paper and I will not go too far into this direction of research. However, I will briefly describe the differences between modeling relational databases and NoSQL databases. As far as data storage is concerned, relational database stores data in two-dimensional tables that consist of rows and columns, and each row has its primary key so that it can be identified in the table. There are also external keys that are used to connect to other tables, and it is also possible to perform various mathematical operations over relations. Precisely because of this structure, database knows what data is stored in it and it is possible to run complex queries over them. Relational databases are vertically scalable, that is, they work best on one computer. On the other hand, NoSQL databases can be classified as follows (Tivanovac, 2016):

- Aggregate Oriented Databases (Key - Value, Column - Family and Document databases) that have horizontal scalability (can be used on multiple computers), and where aggregate is a group of related data which is managed as a whole, so when collecting data from a table, it is enough to only retrieve that aggregate.
- Graph databases that do not use aggregates but have a query language that enables navigation across graph, i.e. nodes. Such databases work better on one server.

Another difference (Gačić, 2017) is that relational databases have uniform data, i.e. all entities within one relation must have the same attributes. Since there is a schema and structure of data, it is necessary to respect it. By contrast, NoSQL databases do not require a data scheme and offer a wide variety of data models that are suitable for storing diverse structures. The biggest problem that arises because of that is the inability to standardize the NoSQL system. Other than that, relational databases only support the storage of text fields, dates, binary strings, and numbers, which is not enough, as there is an increasing need to save entire documents, multimedia content, geographic data and the like, and this problem is solved by NoSQL systems.

2. CLASSIFICATION OF NOSQL DATABASES

There are four basic types of NoSQL systems that can be divided according to the data model. Below are their names and several prototypes (George, 2013):

- Key - value: Riak, Dynamo, Redis, Cache
- Column - family: Cassandra, HBase, Accumulo
- Document: MongoDB, Couchbase, SimpleDB
- Graph: Neo4j, GraphDB, Allegro, Virtuoso

The following subchapters of this review paper will present the four main types of NoSQL databases in more detail and will present various types of simple queries that can be made over them.

2. 1 Key – Value

The key – value databases are simple databases that store data in the form of pairs, where the first component is the key, and the other one is a value associated with that key. The key consists of a string of characters, or a string which is representing a unique code, while the value may appear in any type of data, even in the form of image files, audio files, documents, etc. This value is also called the BLOB (Binary Large Object), which denotes the binary data group stored in the database as one entity. Such a data model can be compared with the dictionary.

There are two properties that apply to these databases, which are:

- All keys must be different, i.e. each key must be unique
- Queries are based solely on the keys, and there is no search option based on the value within the database, which is one of the disadvantages of these databases

Such databases have great performance but are not practical for complex data models. As already mentioned, they do not have their own query language, only the operations of finding, adding and removing pairs based on the selected key, that is, the GET, PUT, and DELETE operations, which have one of the keys as a parameter to execute them (Stojanović, 2016). With the GET operation we get the value assigned to the selected key, with PUT we can add a new pair, key - value, or assign a new value to the already existing key, and using the DELETE operation we remove the selected pair from the database. These operations are what makes a simple API (Application Program Interface).

One of the prototypes of key – value databases is Riak database. It is designed to work in the Internet environment and the commands must be sent using the HTTP protocol. Queries are executed solely on the key that can be generated using the algorithm, provided to the user by an ID or an e-mail address, or exported from the timestamp. The method and ability to query depends on how the key is shaped.

Because of the simplicity, which provides a quick and scalable recognition of the values required for application tasks to the key – value databases, they can be applied in situations where we want to store a user's session, a user's shopping cart, or get details like his favorite products. Analyzing the collected data can improve the work of the entire business organization. That is why Amazon uses its own Dynamo system (Moniruzzaaman, Hossain, 2013).

2. 2 Column – family

Column – family databases are databases where tables are used, i.e. values are stored in rows and columns. These databases use row and column identifiers as the keys which are being used for querying. They can also be compared with spreadsheets. Cells within the database can contain any value, just like the key - value databases. The advantages of these databases are that they have a good benchmark and they store variants of large binary objects - the BLOB, into one large table. Columns that contain related data are grouped together in the column family, resulting in the name of these databases, and thus provides good performance for certain types of searches. On the other hand, some of the disadvantages are that the content of these large binary data can not be queried, and the design of rows and columns is quite critical (Marić, 2015; McCreary, 2014).

One of the examples of column – family databases is Cassandra database. It is written in Java and is described as a combination of Dynamo and Bigtable. In Cassandra, the data is stored in a column in the form of a pair consisting of the name that represents the key and its corresponding value. To avoid conflicts while writing, columns are always stored with a timestamp that is also retrieved with the data while reading.

When we want to run database queries, first we need to specify the keyspace with the entire column family that we will use, which in this example (Marić, 2015) represents "use authors". Then, we are ready to query. Cassandra has its own query language, CQL (Cassandra Query Language), which is quite similar to SQL commands, and therefore this kind of database is ideal for users familiar with the work of relational databases. One of the differences is that CQL does not support join operations and subqueries. Also, one of the advantages of the column - family databases is that they can access individual columns independently of the other columns in the table and therefore the entire table does not have to load. This results in significant memory savings. For example, if a table holds 25 GB of space and we only use one specific column which holds 1 GB of memory, we only need 1 GB for querying that column (Stojanović, 2016).

Example of creating column – family database and entering data:

```
CREATE COLUMNFAMILY Books (
        KEY varchar PRIMARY KEY,
        title varchar,
        author varchar);
```

```
INSERT INTO Books (KEY, title, author)
        VALUES('thecall',
        'The Call of the Wild',
        'Jack London');
```

Example of a query which retrieves information about the book title and name of the author:

```
SELECT title, author FROM Books
```

2. 3 Document

Document databases are the most popular type of NoSQL databases and they serve to store documents which consist of ordered pairs of key and value. Documents are placed in a collection of documents. The advantage is that they are ideal for searching, they have a flexible structure that has been achieved by nested hierarchies, and the disadvantage is that they are very complex for the imeplementation (McCreary, 2014).

Document databases are used when there is no need to store data in a table with uniform field sizes, and if the database does not contain many relations and normalization. Instead, the data is stored in the form of a document with special characteristics. Such databases have application in content management systems, blog software, etc. (Nayak et al., 2013).

Operations that can be performed on the document databases include: entering or updating keys and documents, retrieving or deleting keys. Documents are only accessed using a key, but unlike the key – value databases, the values in the documents can be indexed and searched, i.e. queries can be made over them.

One of the most popular document databases is MongoDB. It consists of dynamic schemes and uses BSON format, which represents binary JSON documents and which affects the speed and simplicity of data integration. MongoDB supports "Ad hoc" queries, therefore it can retrieve fields, documents, and it supports user-defined JavaScript functions (Tivanovac, 2016).

Just like with other document databases, for MongoDB databases it is possible to query parts of a document without the need to retrieve the entire document. Also, in order to avoid processing huge amounts of data, it is important to index this data so we do not have to search every single document in the collection, but only the one we need (Anić, 2016). The basic search command in MongoDB is:

```
db.collection_name.find().
```

If we wanted to retrieve all the users with the last name „Doe" and living at „Ilica 112", we would write the following phrase and arguments:

```
db.users.find({lastName: 'Doe',

address: 'Ilica 112'}).
```

"Users" in the phrase above represents the name of the collection. If we want to retrieve all users older than 20 years old, we'll write the following query:

db.users.find({age: {$gt: 20}}),

where condition {$gt: 20} is "greater than 20" (Stojanović, 2016). We will also present a more complex query and compare it with the SQL query. Our goal is to retrieve the ID of the book and the date of issue through a specific author's ID within the "Book" collection:

SQL:

SELECT bookID, bookDate FROM

books WHERE authorID = '123abc',

MongoDB:

db.books.find({authorID:'123abc'},

{bookID: 1, bookDate: 1})[1].

2. 4 Graph

Graph databases store simple data with complex interconnections which are shown in the graph. They consist of nodes and links between them, and are useful when we are more interested in data relations than data itself. Nodes and links contain their own properties, and links may also have a certain direction that is essential when searching for different patterns and interpretation of results. According to (McCreary, 2014) the advantages of graph databases are the ability to quickly search the network and working with publicly-connected data sets, while the disadvantage is the poor scalability in case when the graphs do not fit into RAM.

Graph databases can be applied to various types of applications, such as social networks, software referral systems, bioinformatics, network management, forensic investigations etc. (Nayak et al., 2013).

As an example we will describe the Neo4j graph database. It uses Cypher query language which with its structure reminds of SQL. Queries are usually presented as a graph traversal, i.e. a journey by graph to reach specific information. Cypher is based on the pattern matching concept by which we can search the structures within the graph that we later use for analysis. One example of the sample can be searching for people living in a particular country. The query we use is:

(:Person)-[:LIVES_IN]->(:City)-[:IS_LOCATED_IN]->(:Country),

in which we pass through three nodes, or two links. In the rounded brackets there are nodes which consist of a colon and a name which together represent the node types. Node types provide more efficient search because the system can only focus on them without the need to search for other nodes. Link types are located within square brackets, and are surrounded by "-" and "->" in the case of directed relations or "-" and "-" in the case of undirected relations (Stojanović, 2016).

Queries may also consist of multiple clauses, as in the following example:

MATCH(x: Message {sent: 'George'})

RETURN x,

where "Message" is a node which we retrieve together with the data in it, which in this case is the value "George". Using the MATCH clause we find nodes, and with the RETURN clause we specify what needs to be retrieved by that query.

When we want to find the shortest way between two nodes to find the data linking them, we use the following query:

MATCH p = shortestPath(

(AneA:Person {name: 'Ane Johnson'})-[*]-(meg:Person {name: 'Billy Crowe'}))

RETURN p,

in which we use the shortestPath function to get the shortest route from Ane to Billy (Janković, 2015).

Table 1 shows a comparison of the Neo4j graph database with the relational database in the friend-to-friend type 5 data query example, using a set of 1 000 000 people. We notice how with relational database we spend much more time to search for data and how it increases parallel with increasing the depth, while with Neo4j we see only a minor difference in each additional level of request (Stojanović, 2016).

Table 1. Comparison of Neo4j with relational database

Depth	Relational Database	Neo4j	Number of returned types
2	0,016 s	0,01 s	~ 2500
3	30,267 s	0,168 s	~ 110 000
4	1545,505 s	1,359 s	~ 600 000
5	Unfinished	2,132 s	~ 800 000

Source: Stojanović (2016)

2. 5 Overview of differences

Table 2. Differences between NoSQL data models

NoSQL databases				
	Key – Value (Riak)	Column – family (Cassandra)	Document (MongoDB)	Graph (Neo4j)
Query language	HTTP, JavaScript, REST, Erlang	API calls, CQL, Thrift	Volatile memory, File System	API calls, REST, SparQL, Cypher, Tinknerpop Gremlin
Protocol	HTTP, REST	Thrift & custom binary CQL3	Custom, binary (BSON)	HTTP/REST embedding in Java
Execution of queries	Solely on the key	Similar to SQL commands, but without join operations and subqueries	Ad hoc queries, user-defined JavaScript functions	Graph taversal, pattern matching
Programming language	Erlang	Java	C++	Java
Best use	When we want to store a user's session, a user's shopping cart...	When data we need to store doesn't fit on server	When we need dynamic queries, good performance on a big database and when data changes a lot (content management systems, blog software...)	When we are more interested in data relations than data itself (social networks, software referral systems, bioinformatics...)

Source: Moniruzzaman (2013)

Riak is an open source database developed using Erlang. It has a flexible data schema, offers high availability, tolerance and persistence, but it should be avoided in case of highly centralized data storage project where the data has fixed structure. Queries are based solely on the key.

Cassandra was developed using Java and it uses concepts of key – value stores and column – family stores. It also offers high availability, tolerance, persistance and high scalability. The disadvantage is that it takes more time to read data than to write it. It has its own query language CQL which is similar to SQL.

MongoDB, a document database, was developed using C++. It has a high performance and besides consistency, just like Riak and Cassandra it also offers tolerance and persistence. In addition, it provides aggregation, ad hoc queries, indexing etc. BSON format, the format in which the documents are stored, is very efficient both in storage space and scan speen. One disadvantage is that indexing takes up a lot of ram.

Neo4j, developed by Neo Tehnology using Java, is a high performance graph database. It provides a flexible network structure and just like Cassandra it offers high scalability. It usess Cypher query language and queries are presented as a graph traversal. It must be avoided if there are no relationships among the data. (Nayak et al., 2013) (Moniruzzaaman, Hossain, 2013) (Oussous et al., 2018)

3. PRACTICAL EXAMPLE

In this chapter, the focus will be on a selected NoSQL database, which in this case is the MongoDB document database. The task of this review paper was to create documents, fill them with their own data and conduct various types of queries on that data.

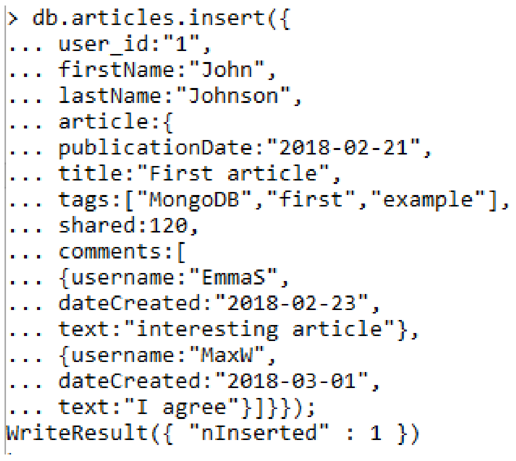
After installing and running MongoDB in the Command prompt, the first step was to create a collection within which we would create the documents. With command

db.createCollection('articles')

we've created a collection named "Articles", and we've started entering the data.

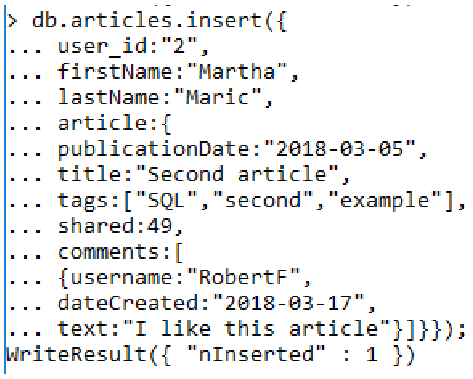
In Figure 1 and Figure 2 we can notice how we entered information for two different articles. We did this by nesting multiple documents within a single document. The article consists of authors, information about the article itself, comments and information about the comments. Both the tags and comments save their values into the fields. With the message "WriteResult ({" nInserted ": 1})" it was verified that we had successfully entered the data inside the document.

Figure 1. Data entry for the first article



```
> db.articles.insert({
... user_id:"1",
... firstName:"John",
... lastName:"Johnson",
... article:{
... publicationDate:"2018-02-21",
... title:"First article",
... tags:["MongoDB","first","example"],
... shared:120,
... comments:[
... {username:"EmmaS",
... dateCreated:"2018-02-23",
... text:"interesting article"},
... {username:"MaxW",
... dateCreated:"2018-03-01",
... text:"I agree"}]}});
WriteResult({ "nInserted" : 1 })
```

Figure 2. Data entry for the second article



```
> db.articles.insert({
... user_id:"2",
... firstName:"Martha",
... lastName:"Maric",
... article:{
... publicationDate:"2018-03-05",
... title:"Second article",
... tags:["SQL","second","example"],
... shared:49,
... comments:[
... {username:"RobertF",
... dateCreated:"2018-03-17",
... text:"I like this article"}]}});
WriteResult({ "nInserted" : 1 })
```

The database is now ready for various queries. To see what's in the collection "Articles" we enter:

db.articles.find();,

and if we would like to get only the first document from the collection we write:

db.articles.findOne().

If we want to see all the information about the comments related to the article whose author has ID = 1, we'll include the following query:

db.articles.find({user_id:'1'},

{'article.comments':1}).pretty(),

where the “pretty()” function allows the results to be displayed in a shapely, more clear way. With the next command we will get the first names and last names of all the authors within the collection:

db.articles.find({}, {firstName:1, lastName:1}).pretty().

Next, to get only the first name of the first author, we will enter:

db.articles.find({user_id:'1'},{'firstName':1}),

or just the last name of the second author:

db.articles.find({user_id:'2'},{'lastName':1}).

If we want to show all information about articles within the collection, descending by user ID, we'll make the following query:

db.articles.find({}).sort({user_id:-1}).

If we are interested in how many articles are in the collection we will use the function “count()”:

db.articles.count(),

and as feedback we will get the number of articles. With the query:

db.articles.find({'article.title': 'Second article'})

all information about the article whose title is “Second Article” will be displayed. If we are interested in an article that has been shared less than 50 times, we will use the command:

db.articles.find({'shared': {$lt:50}}).

The “$lt” tag replaces the operator “less than”.

Using the next query we can retrieve the publication date and title of the article which as a tag has the expression “SQL”:

db.articles.find({'article.tags':'SQL'},

{'article.publicationDate':1,

'article.title':1}).pretty()

In a situation where we need certain information, such as a list of all tags and a title of the article that has been shared more than 100 times, the following query will be valid:

db.articles.find({'article.shared': {$gt:100}},

{'article.tags':1, 'article.title':1}).pretty().

The last command used in this example is the one that retrieves the text of all the comments which were written for the article by author whose ID is 1.

db.articles.find({user.id:'1'},

{'article.comments.text':1}).pretty()

4. CONCLUSION AND FUTURE WORK

This review paper was written to provide a better overview of the NoSQL databases and their models and to help people learn a bit more about them and decide which data model suits them best. NoSQL does not have the standard query language, and each of these four basic database management systems has developed its own query language, way of structuring and providing data. NoSQL databases do not only differ in their provided data model, they also differ in the richness of their offered query functionalities. The current state of the NoSQL environment can be compared to the time before Codds rules for relational databases and SQL. Databases developed in the last years are very heterogeneous - they differ in their data model (if it even exists), query languages and APIs. Therefore, some research is done in order to achieve a wider adoption of NoSQL systems by developing standardized query languages (Meijer, 2011) for some of these stores. Up to now developers still have to cope with the specific characteristics of each NoSQL store.

Since no common query language is available, every NoSQL database differs in its supported query feature set. Riak executes queries solely over the key, Cassandra supports its own CQL query language, MongoDB uses JavaScript as a language to work with the database, and Neo4j uses Cypher query language.

In the future work we will make own tests based on NoSQL databases, which will include testing and comparing performance between different NoSQL databases and their models, as well as flexibility of their data schema, availability, tolerance, persistence, consistency and scalability.

"Every job has its tool" is the propagated ideology of the NoSQL community, because every NoSQL database is specialized on certain use cases. We can conclude that for this reason, NoSQL systems are not ideal because there is a need for a common query language such as SQL, which could be used for all NoSQL databases, and thus make it easier for users to work with them. If such common language appears one day, it is likely that the number of users will increase, and one of the assumptions is that such databases may even become more popular than relational databases.

REFERENCES

Anić, V. (2016) NoSQL baza podataka računalnih komponenti, [online], Available at: https://zir.nsk.hr/islandora/object/etfos:850 [Accessed 4 May 2018]

Gačić, J. (2017) NoSQL baze podataka, [online], Available at: https://zir.nsk.hr/islandora/object/pmf:3234 [Accessed 20 February 2019]

George, Dr. S. (2013) NoSQL – NotOnly SQL, International Journal of Enterprise Computing and Business Systems, Vol. 2 [online], Available at: http://www.ijecbs.com/July2013/3.pdf [Accessed 3 May 2018]

Janković, O. (2015) NoSQL graf baza podataka: od domena do modela preko upita, Vol. 14, [online], Available at: http://infoteh.etf.unssa.rs.ba/zbornik/2015/radovi/RSS-3/RSS-3-2.pdf [Accessed 3 May 2018]

Marić, P. K. (2015) NoSQL baze podataka, [online], Available at http://darhiv.ffzg.unizg.hr/id/eprint/5773/ [Accessed 3 May 2018]

McCreary, D. (2014) Making Sense of NoSQL, Kelly – McCreary & Associates, LLC, [online], Available at: http://macc.foxia.com/files/macc/files/macc_mccreary.pdf [Accessed 3 May 2018]

Meijer, E., Bierman, G. (2011) A Co-Relational Model of Data for Large Shared Data Banks, Available at: Communications of the ACM vol. 54, pp. 49-58. https://doi.org/10.1145/1924421.1924436

Moniruzzaman, A. B. M., Hossain, S. A. (2013) NoSQL Database: New Era of Databases for Big data Analytics – Classification, Characteristics and Comparison, [online], Available at https://arxiv.org/abs/1307.0191 [Accessed 4 May 2018]

Nayak, A., Poriya, A., Poojary, D. (2013) Type of NOSQL Databases and its Comparison with Relational Databases, Vol. 5, [online], Available at: https://research.ijais.org/volume5/number4/ijais12-450888.pdf [Accessed 4 May 2018]

Oussous, A., Benjelloun, F. Z., Lahcen, A. A., Belfkih, S. (2018) Comparison and Classification of NoSQL Databases for Big Data, [online], Available at: https://www.researchgate.net/profile/Ayoub_Ait_Lahcen/publication/278963532_Comparison_and_Classification_of_NoSQL_Databases_for_Big_Data/links/55880d3c08ae1dfa49d211f3/Comparison-and-Classification-of-NoSQL-Databases-for-Big-Data.pdf [Accessed 20 February 2019]

Stojanović, A. (2016) Osvrt na NoSQL baze podataka – četiri osnovne tehnologije, [online], Available at: https://hrcak.srce.hr/192140 [Accessed 3 May 2018]

Tivanovac, M. (2016) Modeliranje ne relacijskih baza podataka, [online], Available at: https://zir.nsk.hr/islandora/object/etfos%3A970 [Accessed 4 May 2018]

Pregledni rad
https://doi.org/10.31784/zvr.7.1.9
Datum primitka rada: 14. 1. 2019.
Datum prihvaćanja rada: 25. 2. 2019.

IZVOĐENJE UPITA NAD PODACIMA U NOSQL BAZAMA PODATAKA

Andrea Babić
Univ. bacc. inf., student, Sveučilište u Rijeci, Odjel za informatiku, Radmile Matejčić 2, 51 000 Rijeka, Hrvatska; *e-mail*: andreababic051@gmail.com

Danijela Jakšić
Dr. sc., poslijedoktorand, Sveučilište u Rijeci, Odjel za informatiku, Radmile Matejčić 2, 51 000 Rijeka, Hrvatska; *e-mail*: danijela.jaksic@inf.uniri.hr

Patrizia Poščić
Dr. sc., redoviti profesor, Sveučilište u Rijeci, Odjel za informatiku, Radmile Matejčić 2, 51 000 Rijeka, Hrvatska; *e-mail*: patrizia@inf.uniri.hr

SAŽETAK

Zadatak rada bio je istražiti NoSQL baze podataka, fokusirati se na izvođenje upita nad njima, opisati neke od dosadašnjih istraživanja te prikazati izvršavanje tih upita na vlastitom praktičnom primjeru. U uvodu su ukratko predstavljene i opisane NoSQL baze podataka. Prikazano je i nekoliko usporedbi s relacijskom bazom podataka. U idućem poglavlju možemo vidjeti pregled osnovnih NoSQL baza podataka te njihove prototipe. U svakom od sljedećih potpoglavlja detaljnije su opisane vrste NoSQL baza podataka i prikazani razni upiti koji se mogu izvoditi nad njima, a u posljednje poglavlje uključen je i vlastiti praktični primjer izvođenja upita nad jednom od tih baza.

Ključne riječi: *NoSQL, baza podataka, ključ – vrijednost, dokumentne baze, upiti*