A Comparative Study of Databases with Different Methods of Internal Data Management

Cornelia Győrödi
Department of Computer Science and Information Technology, University of Oradea
Oradea, Romania

Alexandra Ştefan
Department of Computer Science and Information Technology, University of Oradea
Oradea, Romania

Robert Győrödi
Department of Computer Science and Information Technology, University of Oradea
Oradea, Romania

Livia Bandici
Faculty of Electrical Engineering and Information Technology, University of Oradea
Oradea, Romania

***Abstract*—The purpose of this paper is to present a comparative study between a non-relational MongoDB database and a relational Microsoft SQL Server database in the case of an unstructured representation of data, in XML or JSON format. We mainly focus our presentation on exploring all the possibilities that each type of database offers us, in the case that the data, which has to be stored, cannot or is not wanted to be normalized. This is a scenario most often found in production when, for the application that is being developed we are extracting unstructured data from social networks or all kinds of different channels that the user might have. The comparative study is based on the creation of a benchmark application developed in C# using Visual Studio 2013, which accesses databases created beforehand with proper optimizations that will be described.**

***Keywords*—MongoDB; Microsoft SQL Server; NoSQL; non-relational database**

I. Introduction

Nowadays, applications must support millions of users simultaneously and be able to handle a huge volume of data. A relational database model has serious limitations when handling huge volume of data. These limitations have led to the development of non-relational databases, also commonly known as NoSQL (Not Only SQL) [10].

The relational database model has a rigid schema which means that a schema must be designed in advance before data had been loaded and all attributes of the schema are uniform for all elements, in the case of missing values, null values are used instead [11]. Relational databases are known for their usefulness in terms of data that can be normalized and data that requires transactional integrity.

Non-relational databases do not store data in tables, the schema is not fixed and have very simple data model, and they can handle unstructured data such as documents, e-mail, multimedia, and social media efficiently as shown in [12].

We often encounter unstructured data in XML or JSON format, which cannot be normalized or normalization is not desired. It is important to know this type of data, when and why we should use a relational data model such as SQL Server database instead of a document-oriented database, such as MongoDB and what are the advantages and disadvantages.

It is necessary to do a careful analysis and consider main factors as the amount of data, the flexibility of schema, the budget, the amount of transactions that would be made, when choosing the data model for the application [13].

Generally, for smaller and medium applications, a relational database would be advisable and for big applications, that use and manipulate large quantities of data, a non-relational database is more appropriate [13].

In the first part of this paper, we will be presenting some information about SQL Server and the XML data type in SQL Server, and then we will continue with MongoDB and BSON data type. These will constitute a general knowledge that one needs to have in order to understand the logic behind each database type. In the second part, we will focus on experiments conducted with the help of the benchmark application in order to determine which of these two types of databases is more efficient and in what case. We will also present the experimental results and comparative study with the scenarios in which these results have an impact.

II. Unstructured Representation of Data in Microsoft SQL Server

Microsoft SQL Server is a relational database management system and it is one of the most popular systems used. We can securely say that, at the moment, the database market is dominated by systems that support the relational data model [1].

E. F. Codd proposed the relational model in 1970; D. D. Chamberlin and others from the IBM research lab from San Jose have developed the language that we now call SQL (Structured Query Language) [1]. There are many database management systems that have incorporated SQL and one of them is obviously the Microsoft SQL Server.

This work was performed through the Partnerships Program in priority areas, PN-II-PT-PCCA-2013-4-2225 - No. 170/2014 developed with the support of MEN - UEFISCDI, “Electromagnetic methods to improve processes wine”.

Database management systems, such as Microsoft SQL Server, enjoy a high popularity precisely because they are easy to use and databases are easy to create. Microsoft SQL Server offers reliable transaction processing which is why so many choose this database management system.

Keeping the integrity of our data is often times the most important thing and Microsoft SQL Server has great support for it.

A. The XML Data Type

If the data is structured then our best choice for storing this data is the relational model. If the data is unstructured or semi-structured, we have several options. One would be using a NoSQL database and we will describe this possibility in the next chapter. Another option is using XML data type and this is particularly a good choice if unstructured data are tied in some way to structured data that is already stored in relational database. In this way, we will get a model that is independent from the platform and can be ported easily as shown in [2].

There are many reasons for choosing XML, some of the best, according to Microsoft, are shown in [2]:

- we don't have big quantities of data or the structure of our data is not known at the moment, we maybe have to take into consideration that our data structure might change in the future;
- we have recursive data or the entities don't have references among themselves;
- we have to follow a specific order in our data;
- we hardly ever need to update the whole entity at once, we want to update specific parts of it, change the structure or just simply query.

We have two options of storing XML data: either store it in SQL Server database and use its native XML features or choose to manage it in the file system. We choose considering some of the best reasons as shown in [2]:

- we need transactional integrity, so the most important reason would be that we need to share, query and modify the XML data in an efficient and transacted way;
- we want our relational data to work with or use parts of our XML data;
- we need support for querying and updating data, especially for a cross-domain application;
- we want indexing for an efficient way of querying the data that is stored in XML format.

For choosing to store our XML data in an SQL Server database, one has the option to store it in varchar(MAX), but as we want to take full advantage of what Microsoft SQL Server can offer us, we are going to talk about storing XML in the *xml data type*. We will also keep in mind that storing XML in the *xml data type* is slower due to the validation that happens in the background, but this can give the advantage of having all kinds of information about the specific order in the document, about attribute and element values.

In order to obtain the results from the experiments we used the hybrid model. The hybrid model is a combination of relational and *xml data type* columns [2]. The choice was made in order for the performance to be considerably better.

III. Unstructured Representation of Data in MongoDB

MongoDB is a document-oriented, NoSQL database. NoSQL, or Not Only SQL, is an approach of managing data and designing databases, which is most useful in the case that we have big quantities of data [3]. NoSQL databases provide you with ways of storing and retrieving the data that is not modelled as the relational databases are modelled. Mainly, NoSQL databases are designed to allow us insertion of data for which we do not have a predefined schema as the structure of our data is not set.

A database like MongoDB does not a have the concept of a "row"; instead, we have a more flexible model called a "document" [3]. The format in which the documents are stored is called BSON which comes from binary JSON and which offers us a binary representation of the JSON documents.

We have an easy way of modifying the structure of our data as MongoDB does not restrict to certain types or sizes, without having a predefined schema, we can experiment with modelling our data and choose the best option according to the needs of the application [3].

Often times the most challenging thing that developers are confronted with is the ever-growing amount of data that our applications deal with. As we need to store this data, the problem of scaling arises.

There are two choices when it comes to scaling: either we can scale up or we can scale out. Scaling up implies upgrading the machine we already have, basically adding more resources, while scaling out is getting our data spread across multiple machines [3]. Scaling up is generally more expensive and the physical limitation will inevitably be reached at some point [3]. Scaling out will come with a requirement of a bigger effort in order to administer the multiple machines, but it is generally less expensive and easier to scale [3].

When wanting to scale out a relational database we have to understand that it is generally not an easy problem to solve. However, MongoDB was made precisely with this process in mind. Being document oriented makes it easy to split the data and MongoDB figures out how to spread the data across the newly added machines [3].

The important thing to note about MongoDB is that while it has many features that facilitate CRUD operations, some features that we most often use in relational databases like joins, are not possible in MongoDB. We have a way of simulating this type of operation, which will be presented later on in this paper.

IV. Experimental Setup

Our working scenario is when we have data that cannot be normalized, but is still connected in some way to existing data in the relational database. We have a choice between using the hybrid model for SQL Server against storing the data in

MongoDB and just retrieving it from there. For the SQL Server, we will store an ID and an *xml data type* field. For MongoDB, the data will be stored as documents, which together will form a collection.

In order to have the fairest comparison we created indexes designed to ensure the optimal performance. As a result, for the *xml data type* we created the following indexes:

- primary XML index – this is the most important index that we created as this one indexes all the XML tags, values and paths [4]. According to Microsoft, for the creation of this index we need a clustered index on the primary key of the table that contains *the xml data type* column as SQL Server will use the primary key to correlate rows in the primary XML index with rows in our table [4];
- secondary XML index – in order to be able to create two types of secondary indexes we needed a primary XML index. These are the types of secondary indexes created:
    - path – used for queries that specify path expressions because it makes searching faster [5];
    - value – used for value based queries, an example would be searching for a string [5].
- full-text index on a XML column – according to Microsoft, it indexes the content of the XML values, but ignores the XML mark-up [6].

MongoDB has a default index on the *_id* field, so if now of creation we do not set it, this *_id* will be automatically set. Like the concept of primary key in SQL, this *_id* prevents the introduction of two *_id* values that are the same and is unique [7]. These are the following indexes created for the MongoDB database:

- single field – it is either an ascending or descending index specified by the user on single field of the document [7];
- compound index – it is an index on multiple fields from the document and the order in which you specify the fields is very important as MongoDB will sort after the first field and then it will sort within each value of the first field by the second field specified [7];
- text index – it is an index that supports running text search queries in a string content. One can specify any field that has a string as a value or an array of strings, according to MongoDB [8].

In order to run the experiments, we created a benchmark application using C# and Visual Studio 2013 as an IDE (Integrated Development Environment). Using the repository pattern, we created two repositories for each database. For the execution of the SQL commands, we used *SqlCommand* from *SqlClient* that is the .NET Framework Data Provider for SQL Server and for MongoDB we used the *.NET MongoDB Driver*. Both provide asynchronous workflows.

The architecture of the computer used to run the experiments:

Operating System	Windows 10 Pro
Processor	Intel(R) Core(TM) i5-4200M CPU @ 2.50 GHz
Installed memory (RAM)	4.00 GB
Disk	SSD Crucial MX100 256GB

V. Experimental Results

A. *Experiment 1*

The first experiment consists of populating the two databases with 100.000 entries. The chart shown in Figure 1 presents the results of the experiment:

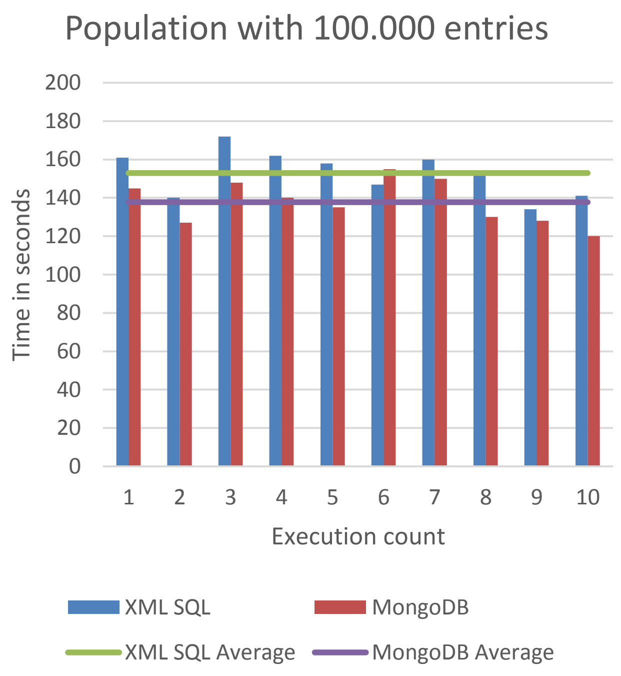


Fig. 1. The results of populating with 100.000 entries

We can easily see the implications of inserting an already considerable amount of data, both in SQL Server and in MongoDB. MongoDB is faster, usually being tens of seconds faster than SQL Server. The difference occurs also because of the XML validation done by SQL Server. The *xml data type* ensures us that each XML instance is correctly formed and this process slows down the insertion.

As it can be seen in Figure 1, MongoDB is faster than SQL Server in 9 out of 10 cases. The method used for insertion is similar in order to not give an advantage through implementation.

B. *Experiment 2*

The purpose of this experiment was to search by a randomly generated *ID* 1.000 times on each execution. As previously described the *ID* on the SQL Server database is a primary key and on the MongoDB database *_id* field has a default index on it.

The results of this experiment are pointing to the conclusion that searching by the *ID* field on which have a primary key or default index on it is yielding better results in SQL Server than in MongoDB. SQL Server is efficient and fast in these types of operations as it shown in Figure 2.

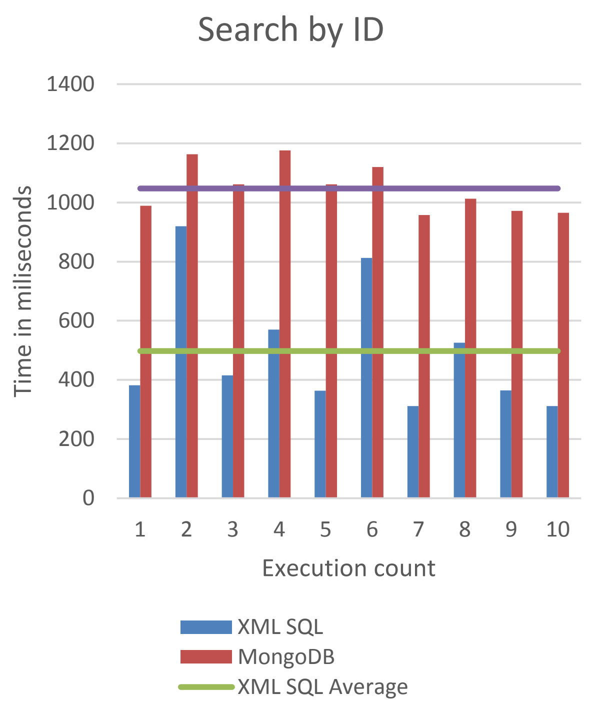


Fig. 2. The results of searching by ID

C. *Experiment 3*

The third experiment consisted in searching for a random string 1.000 times at every execution. In this experiment, we aim to test the full-text index and the text index that we set for each database type. The results, as shown in Figure 3, are rather dramatic as the difference between the two database types are quite big. MongoDB finds it simply easier and more efficient to search for particular occurrences of a string.

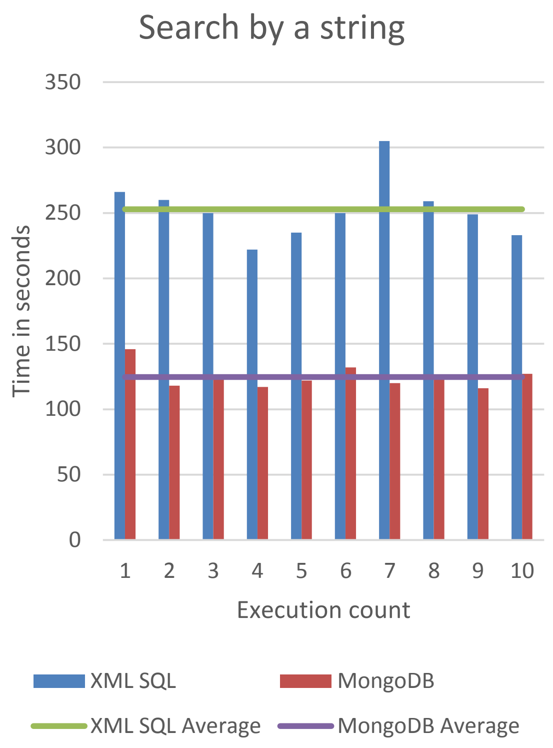


Fig. 3. The results of searching by string

D. *Experiment 4*

The purpose of this experiment is to update a field that has a randomly generated *ID* 1.000 times.

MongoDB clearly dominates in these types of operations, as we can see in Fig. 4, the difference is yet again major and in favour of our NoSQL database. MongoDB enables superior performance as querying in the XML using the *xml data type* methods, but is not nearly as fast as MongoDB's easy way of looking up the document by its ID and updating its field.

This is the method chosen to update a field in the SQL Server database:

```
UPDATE Post SET Xml.modify('replace value of
(/post/category/text())[1] with
(\"UpdatedCategory\")')
WHERE Id = @Id
```

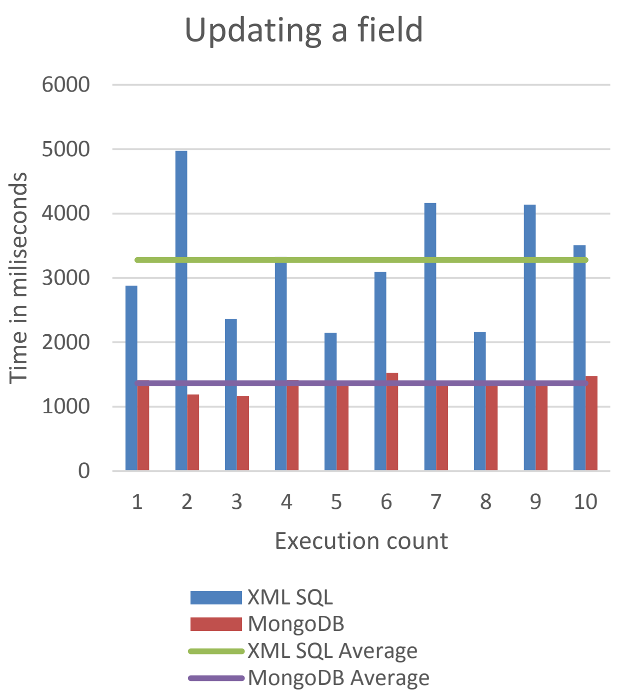


Fig. 4. The results of updating a field

E. *Experiment 5*

The aim of this experiment is to compare the execution speed of join operations. MongoDB does not support join operations as it goes against the concept of data getting denormalized [9]. The addition of redundant data reduces the need for join operations. However, in certain scenarios we do not want to keep redundant data in our documents, and for this particular need MongoDB offers two solutions:

- manual references –meaning that we ought to have a field that will store the primary key of the document where the related data resides [9];
- *DBRefs* – this is a reference between two documents using the *_id* field, the name of the collection and optionally the name of the database [9].

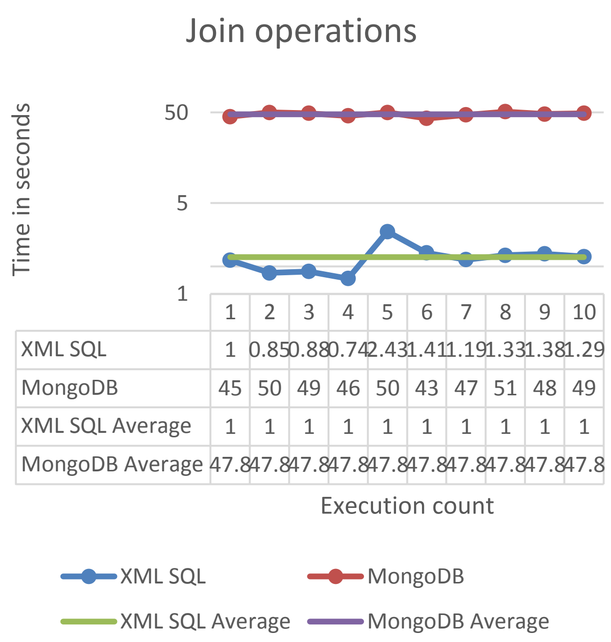
In this particular scenario, we know in which database the collection resides and we have all the information we need in our application meaning that *DBRef* does not give us any advantage over the manual reference. With that in mind, we chose to use the manual reference and to create an index on the reference field.

In the setup phase of our SQL Server database, we have created two tables – one that holds the posts, stores an *ID*, and has an *xml data type* field, which stores the XML and one table that holds the comments, which stores the *post_id* and some random content. We mirrored this in MongoDB by adding a comments document. It is necessary for us to create an index on the foreign key field *post_id* as this will speed up the operation drastically and will give fairness as we add an index on the *post_id* reference field from the MongoDB database as well.

We added, for each database, between 10 and 100 comments for 25,000 posts.

In our experiment, we made 1,000 join operations using the *ID* field which is, as previously mentioned, primary key in the SQL Server database and default *_id* index in MongoDB. In MongoDB's case, in order to simulate the joint operation we looked first for the post and then for all the comments made for that particular post.

As we can see in Figure 5, the results yielded by the SQL Server database are much better, which is as expected since with proper optimization there is no way that MongoDB can, beat SQL's JOIN. What is notable is that without having an index on the *post_id* foreign key field, SQL Server yielded much worse results than what we can observe here for MongoDB.



	1	2	3	4	5	6	7	8	9	10
XML SQL	1	0.85	0.88	0.74	2.43	1.41	1.19	1.33	1.38	1.29
MongoDB	45	50	49	46	50	43	47	51	48	49
XML SQL Average	1	1	1	1	1	1	1	1	1	1
MongoDB Average	47.8	47.8	47.8	47.8	47.8	47.8	47.8	47.8	47.8	47.8

Fig. 5. The results of join operations

VI. Conclusions

The purpose of all these experiments was to give an answer to the question: when do we use a relational database and when do we use a NoSQL database, like MongoDB?

The answer is not nearly as complex as one might think.

First, we need a proper analysis of the operations that we will do on our database and after that an analysis on the data that we work with. Microsoft SQL Server offers us transactional integrity and speed in JOIN operations, however MongoDB has the superior read and update speed. We must ask ourselves, "Do we have a rigid schema for our data?" Will the structure of our data suffer modifications? How flexible do we need to be when that happens? If our data cannot be normalized, we have to ask ourselves the question, "Does any of the existing data in our relational database relate to our data that cannot be normalized?"

If yes, then is the hybrid model enough? Normalization often requires of us to store the data in many tables and in order for us not lose on performance we need many indexes. The same kind of structure can be modelled in a MongoDB database and in such a way that we completely get rid of the need to use JOIN type operations, which will drastically improve performance and will be considerably faster than any relational database. Identifying the needs of each application is key.

In this particular scenario, having data that cannot be normalized, it is very easy for us to conclude that given huge amount of data, MongoDB will always be the best solution. We can go as far as model our data in a single document, which will always be faster than storing XML in an *xml data type* column in an SQL Server database.

To sum up, while we cannot conclude that smaller amounts of data mean that the hybrid model becomes the best option, we can say that it entirely depends on the needs of the application that is being developed.

References

[1] Matt Levene, George Loizou, "A Guided Tour of Relational Databases and Beyond" Published by Springer-Verlag, London, 1999, pp 1-2.

[2] XML Data Type and Columns (SQL Server) – Available: https://msdn.microsoft.com/en-us/library/hh403385.aspx., accessed January 2016.

[3] Kristina Chodorow, "MongoDB: The Definitive Guide, Second Edition" Published by O'Reilly, May 2013, pp 3-4.

[4] TechNet Library, "Primary XML Index" – Available: https://technet.microsoft.com/en-us/library/bb500237(v=sql.105).aspx, accessed January 2016.

[5] TechNet Library, "Secondary XML Index" – Available: https://technet.microsoft.com/en-us/library/bb522562(v=sql.105).aspx, accessed January 2016.

[6] TechNet Library, "Full-Text Index on an XML Column" – Available: https://technet.microsoft.com/en-us/library/bb522491(v=sql.105).aspx, accessed January 2016.

[7] MongoDB for Giant Ideas, "Index Introduction" https://docs.mongodb.org/manual/core/indexes-introduction/, accessed February 2016.

[8] MongoDB for Giant Ideas, "Text Indexes" – Available: https://docs.mongodb.org/manual/core/index-text/, accessed February 2016.

[9] MongoDB for Giant Ideas, "Database References" – Available: https://docs.mongodb.org/manual/reference/database-references, accessed February 2016.

[10] N. Jatana, S. Puri, M. Ahuja, I. Kathuria, D. Gosain, "A survey and comparison of relational and non-relational databases", International Journal of Engineering Research &Technology (IJERT) ISSN: 2278-0181, Vol 1, Issue 6, August 2012, pp. 1-5.

[11] R. D. Bulos, J. Bonsol, R. Diaz, A. Lazaro, V. Serra,"Comparative analysis of relational and non-relational database models for simple queries in a web-based application", Research Congress 2013, de la Salle University Manila, march 7-9, 2013.

[12] K. Sanobar, M. Vanita, "SQL Support over MongoDB using Metadata", *International Journal of Scientific and Research Publications*, Volume 3, Issue 10, October 2013.

[13] C. Győrödi, R. Győrödi, R. Sotoc, "A Comparative Study of Relational and Non-Relational Database Models in a Web- Based Application", International Journal of Advanced Computer Science and Applications, ISSN : 2158-107X(Print), ISSN : 2156-5570 (Online),Volume 6, Issue 11, 2015, pag. 78-83.